Luis Testing Quarto

Luis Sosa

2024-11-16

```
import pandas as pd
import numpy as np
import os
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import GridSearchCV, train_test_split, cross_val_score, Stratifi
from sklearn.datasets import make_classification
from sklearn.inspection import PartialDependenceDisplay
from xgboost import XGBClassifier


import shap
from catboost import CatBoostClassifier
import matplotlib.pyplot as plt
from joblib import dump, load

pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
pd.options.display.float_format = "{:,.6}".format

# Run H2O later
import h2o
import pandoc
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
from h2o.estimators.gbm import H2OGradientBoostingEstimator
from h2o.estimators import H2ORandomForestEstimator, H2ODecisionTreeEstimator
from h2o.grid.grid_search import H2OGridSearch
from h2o.estimators import H2OKMeansEstimator
from h2o.automl import H2OAutoML
```

```
h2o.init(max_mem_size = "10G") # need more memory
h2o.init() # need more memory
```

```
def firingRate(df):
    return 1-df.isna().sum(axis=0)/df.shape[0]

def uniqueCount(df):
    return df.nunique()

def uniqueRate(df):
    return df.nunique()/df.shape[0]

def firingRateDetailed(df):
    output = []
    firingRate = 1-df.isna().sum(axis=0)/df.shape[0]
    uniqueCount = df.nunique()
    uniqueRate = df.nunique()/df.shape[0]
    for i in df.columns:
        if df[i].dtypes == 'O':
            if df[i].nunique() <= 15:
                output.append(df[i].value_counts(dropna = False, normalize = True))
            else:
                output.append((i+': Categorical Column with '+ str(df[i].nunique()) + ' grou
        elif (df[i].dtypes in ['int8','int16','int32','int64','float32','float64']):
            output.append(df[i].describe())
        else:
            df[i] = df[i].astype(str)
            if df[i].nunique() <= 15:
                output.append(df[i].value_counts(dropna = False, normalize = True))
            else:
                output.append((i+': Categorical Column with '+ str(df[i].nunique()) + ' grou
    final = pd.DataFrame({'Column': df.columns,
                          'firingRate': firingRate,
                          'uniqueCount': uniqueCount,
                          'uniqueRate': uniqueRate,
                          'Detailed_Summary': output})
    return(final)

def liftchart(actual, predicted_scores, loss_amount, nbins = 100):
    df = pd.DataFrame({"actual" : actual,"scores" : predicted_scores, "loss_amount": loss_am
    centile_bins = np.flip(np.array(range(nbins-1, -1, -1))/nbins )#+ np.array([0.1]*nbins)
    score_bins = np.flip(np.quantile(predicted_scores, centile_bins))
```

```
    sum_loss = df[df['actual'] == 1].loss_amount.sum()
    capture_count = []
    loss_total = []
    for i in score_bins:
        df_bin = df[df["scores"] >= i]
        capture_count.append(sum(df_bin["actual"]))
        loss_total.append(sum_loss - sum(df_bin[df_bin["actual"]==1]["loss_amount"]))
    output_df = pd.DataFrame({"centile_bins": np.array([1] * nbins) -  np.array(centile_bins
                              "threshold":score_bins,
                              "number_of_defaults_present": capture_count})
    output_df['cumulative_running_total'] = np.array(capture_count) / sum(df_bin["actual"])
    output_df['default_rate_at_current_bin'] = np.array(capture_count) / len(df)
    #output_df['savings'] = loss_total
    return(output_df)

def compute_class_weights(y):
    # Calculate the class weights
    class_weights = compute_class_weight(
        class_weight="balanced", classes=np.unique(y), y=y
    )

    # Convert the result to a dictionary with class labels as keys
    class_weight_dict = dict(zip(np.unique(y), class_weights))

    # Create an array of the same length as y with the corresponding class weight for each el
    class_weights_array = np.array([class_weight_dict[label] for label in y])

    return class_weights_array
```

```
### h2o.shutdown() #in case you need to shut down
```

Read Results File

```
df = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Visible_1169_Records_Scored_2024
print(df.shape)
```

```
(1169, 3442)
```

```
/var/folders/z5/xxddnyr978bb97d94x1wj3780000gn/T/ipykernel_92556/3848255160.py:1: DtypeWarnin
  df = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Visible_1169_Records_Scored_202
```

```
df.APPROVED.value_counts().to_frame()
```

APPROVED	count
False	958
True	211

```
df.APPROVED.value_counts(normalize = True).to_frame()
```

APPROVED	proportion
False	0.819504
True	0.180496

```
df.STATUS.value_counts().to_frame()
```

STATUS	count
PAID	87
SENT_TO_COLLECTION	48
PAID_OVERDUE	30
FINANCED	20
OVERDUE	15
DUE	8
COLLECTED	3

```
df.STATUS.value_counts().to_frame()
```

Print Evaluation Metrics

Strategy 1

```
strategy1 = h2o.load_model('/Users/luis/Evaluations/Visible/2_Modeling/Strategy1v4_seed_191/
```

```
print("Strategy 1 Train AUC: " + str(strategy1.auc(train = True)))
```

```
Strategy 1 Train AUC: 1.0
```

```
print("Strategy 1 Test AUC: " + str(strategy1.auc(valid = True)))
```

```
Strategy 1 Test AUC: 0.7708333333333334
```

Strategy 2

```
strategy2 = h2o.load_model('/Users/luis/Evaluations/Visible/2_Modeling/Strategy2v5_seed_191_c
```

```
strategy2_train = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Strategy2_Results_fi
    columns = {"Pave_Custom_Score": "Strategy2"})
```

```
/var/folders/z5/xxddnyr978bb97d94x1wj3780000gn/T/ipykernel_58373/1856857068.py:1: DtypeWarnin
  strategy2_train = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Strategy2_Results_
```

```
strategy2_train['label'] = strategy2_train['label'].astype(int)
```

```
strategy2_train.columns[3430:3440]
```

```
Index(['client', 'ID', 'UNDERWRITING_DATE', 'APPROVED', 'AMOUNT', 'DUE_DATE',
       'STATUS', 'weights', 'dataset', 'Pave_Custom_Score'],
      dtype='object')
```

```
print("Strategy 2 Train AUC: " + str(strategy2.auc(train = True)))
```

```
Strategy 2 Train AUC: 0.8472831549268215
```

```
print("Strategy 2 Test AUC: " + str(strategy2.auc(valid = True)))
```

```
Strategy 2 Test AUC: 0.7986111111111112
```

```
train = strategy2_train[strategy2_train['dataset'] == 'train']
print(train.shape) # test size
print(train.label.fillna("NA").value_counts()) # Label Counts
roc_auc_score(train.label, train.Strategy2) # AUC Performance
```

```
(3272, 3440)
label
0    2559
1     713
Name: count, dtype: int64

0.8909267239843754
```

Test AUC confirmed in scoring roc_auc_score (sklearn function)

```
test = df[df['dataset'] == 'test']
print(test.shape) # test size
print(test.label.fillna("NA").value_counts()) # Label Counts
roc_auc_score(test.label, test.Strategy2) # AUC Performance
```

```
(52, 3442)
label
0    36
1    16
Name: count, dtype: int64

0.798611111111111
```

Lift Chart on Test Dataset (we will use this to observe the default rates in top deciles)

```
## Adjustment so the higher the score, the better the applicant
df['Strategy2_Adj'] = df['Strategy2'].apply(lambda t: 1-float(t))
df['loss'] = df['AMOUNT'] ## We will use the CREDIT_AMOUNT as a loss amount (in case of COLL
```

Top Scoring bins and savings on Test Dataset (52 records)

```
test = df[df['dataset'] == 'test']
print(test.shape)
print(test.label.fillna("NA").value_counts())

lc2_test = liftchart(test['label'], test['Strategy2_Adj'], test['loss'], nbins = 100)
lc2_test
```

```
test = df[df['dataset'] == 'test']
print(test.shape)
print(test.label.fillna("NA").value_counts())

lc2_test = liftchart(test['label'], test['Strategy2_Adj'], test['loss'], nbins = 10)
lc2_test
```

```
(52, 3444)
label
0    36
1    16
Name: count, dtype: int64
```

	centile_bins	threshold	number_of_defaults_present	cumulative_running_total	default_rate_at_cu
0	0.9	0.865685	0	0.0	0.0
1	0.8	0.837012	2	0.125	0.0384615
2	0.7	0.796041	2	0.125	0.0384615
3	0.6	0.781759	2	0.125	0.0384615
4	0.5	0.727344	3	0.1875	0.0576923
5	0.4	0.715786	3	0.1875	0.0576923
6	0.3	0.661514	5	0.3125	0.0961538
7	0.2	0.611454	9	0.5625	0.173077
8	0.1	0.545787	12	0.75	0.230769
9	0.0	0.398259	16	1.0	0.307692

Top Scoring bins and savings on Approved Records Only

```
df.STATUS.unique()
```

```
array(['PAID', 'COLLECTED', 'PAID_OVERDUE', 'SENT_TO_COLLECTION', nan,
       'OVERDUE', 'FINANCED', 'DUE'], dtype=object)
```

```
dfa = df[df['APPROVED'] == True ]
print(dfa.shape)
print(dfa.label.fillna("NA").value_counts())

lc2_approvals = liftchart(dfa['label'], dfa['Strategy2_Adj'], dfa['loss'], nbins = 100)
lc2_approvals
#lc2_approvals.sort_values("threshold", ascending = True)
```

Top Scoring bins and savings on All Records (including declines)

```
print(df.shape)
print(df.label.fillna("NA").value_counts())

lc2_overall= liftchart(df['label'], df['Strategy2_Adj'], df['loss'], nbins = 100)
lc2_overall
#lc2_overall.sort_values("centile_bins")
```

Compute new Approval Rates

```
thresholds_lc2 = lc2_overall['threshold'].tolist() # Use thresholds for overal dataset (inclu
```

```
declined_df = df[df["APPROVED"] == False]
approved_df = df[(df["APPROVED"] == True)]

new_approvals_list = []
#new_approvals_income_list = []
existing_approvals_list = []


for i in thresholds_lc2:
    new_approvals = len(
```

```
        declined_df[declined_df["Strategy2_Adj"] >= i]
    )
    new_approvals_list.append(new_approvals)
  #  new_approvals_income = len(
  #      declined_income_df[declined_income_df["pave_custom_score"] >= i]
  #  )
  #  new_approvals_income_list.append(new_approvals_income)

    existing_approvals = len(
        approved_df[approved_df["Strategy2_Adj"] >= i]
    )
    existing_approvals_list.append(existing_approvals)
    print(
        f"{i}: | new approvals: {new_approvals} | existing approvals: {existing_approvals}"
    )
```

```
new_and_existing_approvals = pd.DataFrame({"centile_bins": lc2_approvals['centile_bins'].toli
                           "threshold": thresholds_lc2,
                           "new_approvals - scores above threshold": new_approvals_list,
                             "existing_approvals": existing_approvals_list})
new_and_existing_approvals['Total_Approvals'] = np.array(new_and_existing_approvals['new_appr
                                                + np.array(new_and_existing_approvals['existi
```

```
lc2_final = pd.merge(lc2_test[['centile_bins',
                               'default_rate_at_current_bin']],
                     new_and_existing_approvals[['centile_bins',
                               'new_approvals - scores above threshold',
                               'existing_approvals']], how = 'inner', on = 'centile_bins
lc2_final
```

Compute Margin of Error of Default Rates using Standard Error

```
print("Size of Test Data: " + str(len(test)))
t_star = 2.009 # 51 degrees od freedom
print("t* with 51 (n-1) degrees of freedom: " + str(t_star))

lower_p_hat_list = []
upper_p_hat_list = []
for i in lc2_final.default_rate_at_current_bin.tolist():
```

```
    lower_p_hat_list.append(i - (t_star * (np.sqrt(((i) * (1-i))/len(test)))))
    upper_p_hat_list.append(i + (t_star * (np.sqrt(((i) * (1-i))/len(test)))))

lc2_final['95% CI LowerBound_Default_Rate'] = lower_p_hat_list
lc2_final['95% CI UpperBound_Default_Rate'] = upper_p_hat_list

lc2_final['95% CI LowerBound_Default_Rate'] = np.where(lc2_final['95% CI LowerBound_Default_R
                                                       lc2_final['default_rate_at_current_bin
                                                       lc2_final['95% CI LowerBound_Default_R


lc2_final['95% CI UpperBound_Default_Rate'] = np.where(lc2_final['95% CI UpperBound_Default_R
                                                       0.307692 , 0.307692,
                                                       lc2_final['95% CI UpperBound_Default_R

lc2_final = lc2_final[['centile_bins', '95% CI LowerBound_Default_Rate','95% CI UpperBound_De
       'new_approvals - scores above threshold', 'existing_approvals']]

lc2_final['Total Approvals'] = np.array(lc2_final['new_approvals - scores above threshold'])
                               np.array(lc2_final['existing_approvals'])

lc2_final['Approval Rate at Top Decile'] = np.array(lc2_final['existing_approvals']) /\
                               np.array([1169] * len(lc2_final))



lc2_final
```

Takeaways:

- Visible's current approval rate is 18.0%
- Their default rate is 30.3% (when you exclude "Overdue, Due, and Financed".
- Using our model, if we were to approve the top 40% of applicants in the top of the funnel, we would be able to lower the default capture rate from 30.3% to 10.7% (using a 95% Margin of Error).
- Using this threshold will also increase your current approval rate by 22% (from 18% to 40%).

```
strategy2_train = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Strategy2_Results_fi
    columns = {"Pave_Custom_Score": "Strategy2"})
test = strategy2_train[strategy2_train['dataset'] == 'test']
print(len(test))
```

```
52
```

```
/var/folders/z5/xxddnyr978bb97d94x1wj3780000gn/T/ipykernel_92556/502720196.py:1: DtypeWarning
  strategy2_train = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Strategy2_Results_
```

```
strategy2 = h2o.load_model('/Users/luis/Evaluations/Visible/2_Modeling/Strategy2v5_seed_191_
```

```
varimp_pd = strategy2.varimp(use_pandas = True)
varimp_pd.to_csv("/Users/luis/Evaluations/Visible/2_Modeling/Variable_Importance_drf_grid12_m
varimp_pd.head()
```

	variable	relative_importance	scaled_importance	percentage	
0	primary_account_current_balance_maximum_past_180d	83.6137	1.0	0	
1	primary_account_current_balance_average_past_30d	53.5737	0.640729	0	
2	primary_account_history_days	48.5071	0.580134	0	
3	minimum_current_balance_2d_after_payroll_past_...	45.0793	0.539138	0	
4	atm_fees_count_trend_month_5	44.5181	0.532426	0	

```
test_h2o = h2o.H2OFrame(test[varimp_pd.variable.tolist()+['label']])
```

```
Parse progress: |                                                 | (done) 100%
```

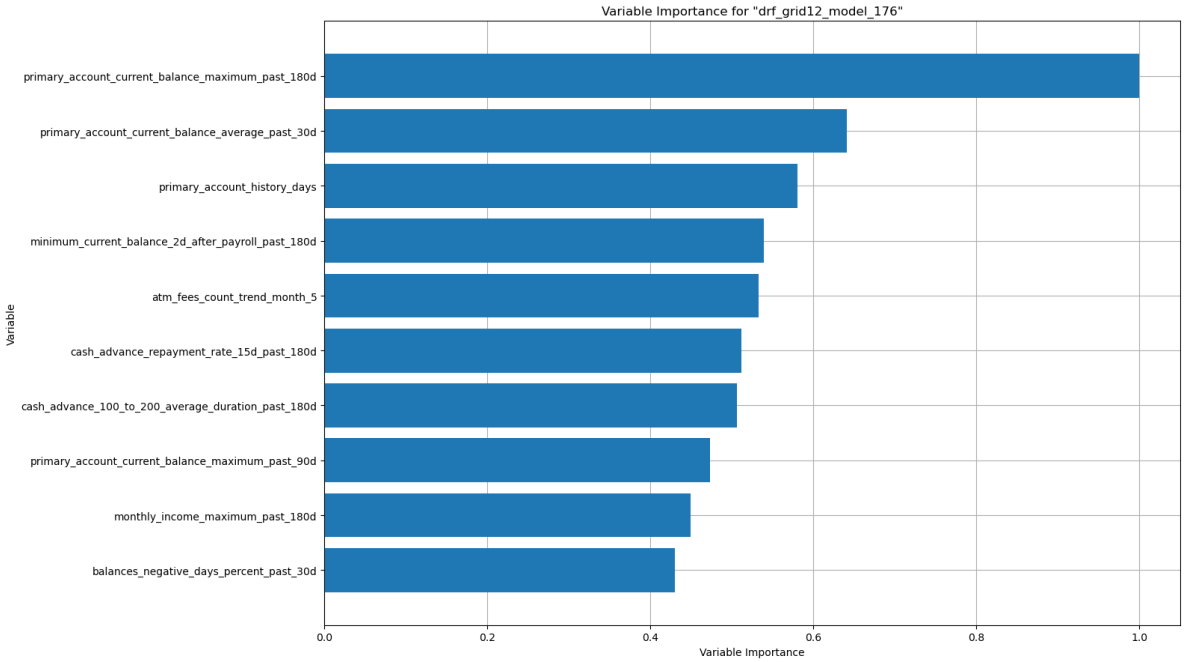
```
# Fix Column Types in h2o
# If column enum has "9 or more" values, turn numeric (to prevent high cardinality in categor
for i in test_h2o.columns:
    if (test_h2o.types[i] == "enum"):
        if len(test_h2o[i].levels()[0]) >= 5:
            test_h2o[i] = test_h2o[i].asnumeric()

    #elif df_h2o.types[i] == "int":
    #    df_h2o[i] = df_h2o[i].asnumeric()

#df_h2o.types
```

```
test_h2o["balances_double_digit_days_percent_past_60d"] = test_h2o["balances_double_digit_da
test_h2o["balances_negative_days_percent_past_30d"] = test_h2o["balances_negative_days_perce
test_h2o["balances_single_digit_days_percent_past_30d"] = test_h2o["balances_single_digit_da
test_h2o["cash_advance_amount_trend_month_6"] = test_h2o["cash_advance_amount_trend_month_6"
test_h2o["insurance_providers_count_trend_month_4"] = test_h2o["insurance_providers_count_tr
test_h2o["investments_count_trend_month_6"] = test_h2o["investments_count_trend_month_6"].as
test_h2o["reversed_loan_payment_amount_trend_month_3"] = test_h2o["reversed_loan_payment_amo
```

```
h2o.explain(strategy2, test_h2o, columns = ['primary_account_current_balance_maximum_past_18
                                                        'primary_account_current_balanc
                                                        'primary_account_history_days',
                                        'minimum_current_balance_2d_after_payroll_past_180d',
                                        'atm_fees_count_trend_month_5',
                                        'cash_advance_repayment_rate_15d_past_180d'],
           include_explanations = ['confusion_matrix',
                                   'varimp',
                                   'pdp',
                                  'varimp_heatmap',
                                  'shap_summary',
                                  'ice','model_correlation_heatmap'])
```

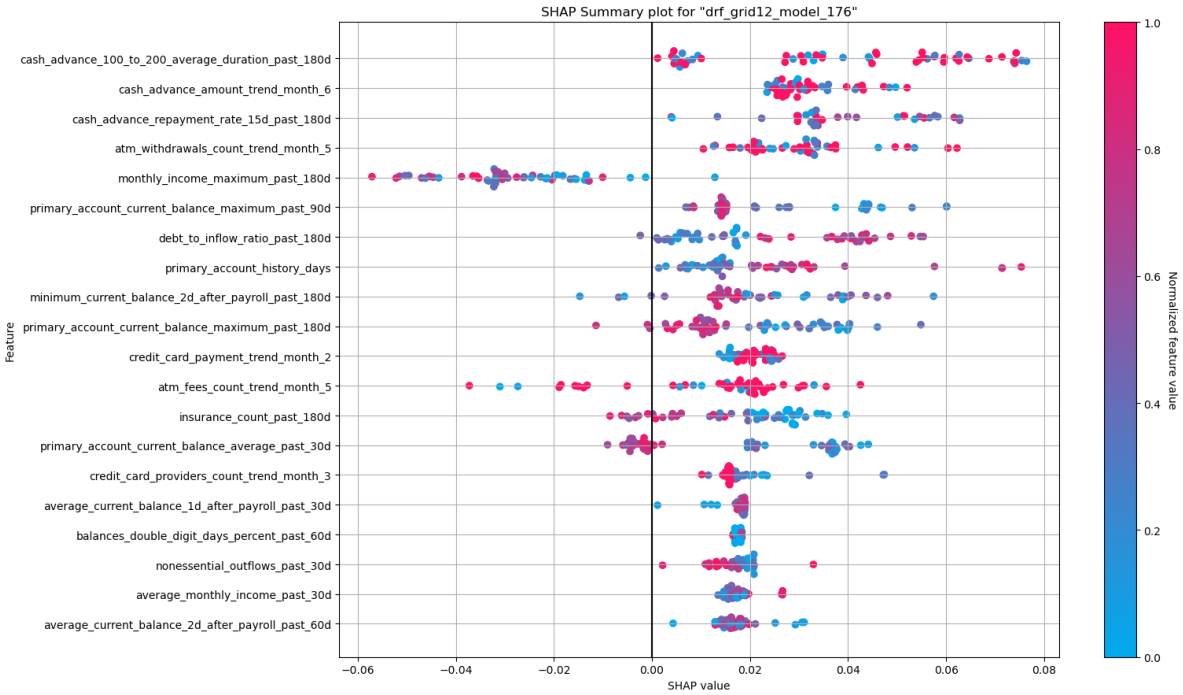
Variable Importance

The variable importance plot shows the relative importance of the most important variables in the model.



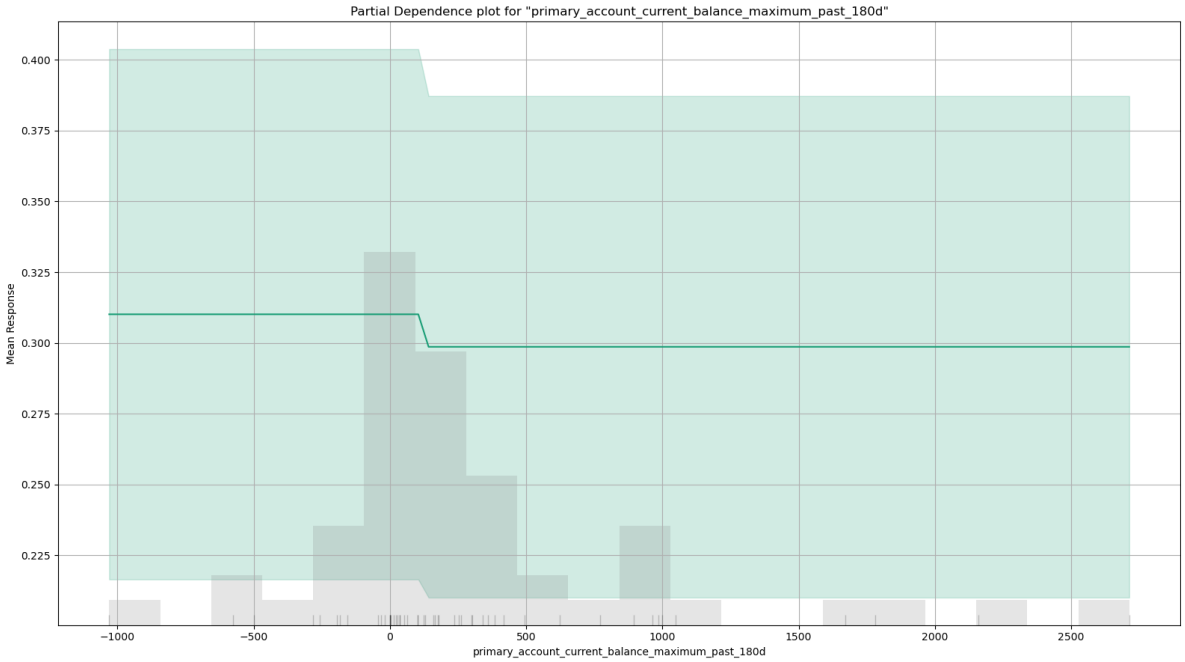
SHAP Summary

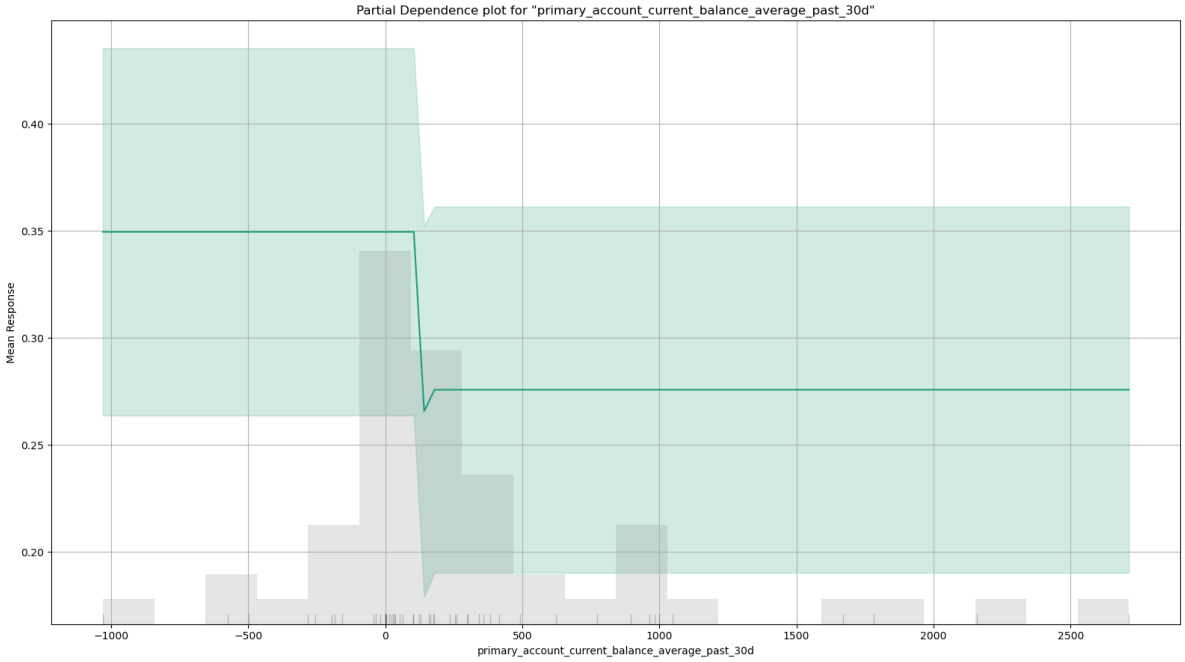
SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.

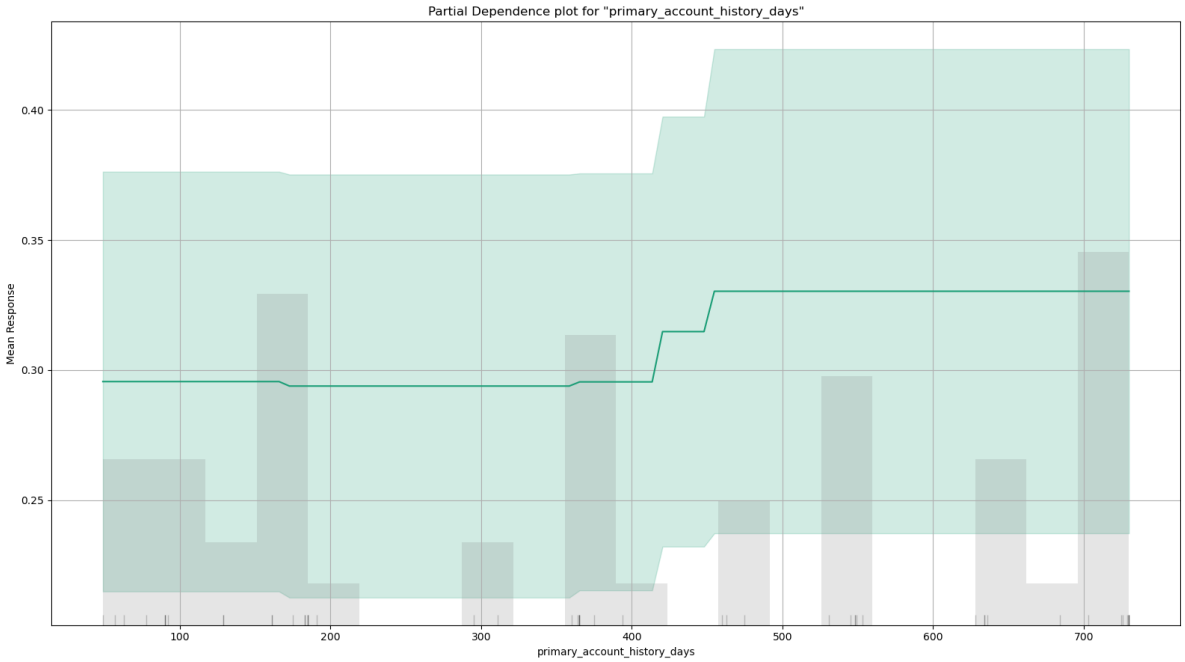


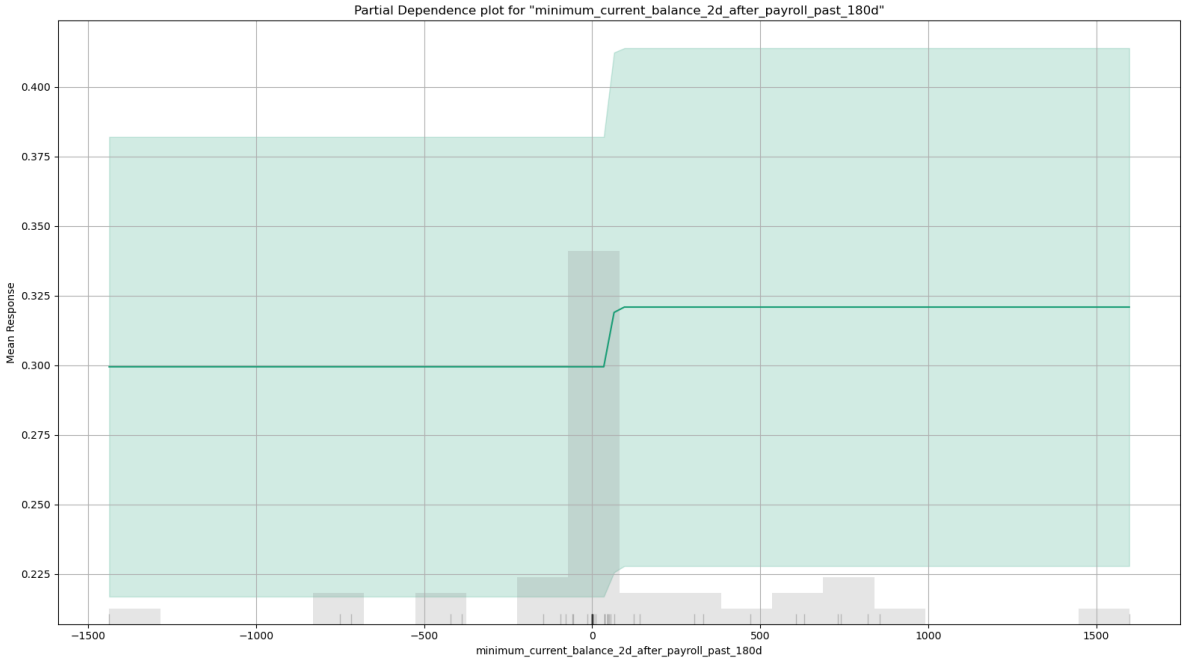
Partial Dependence Plots

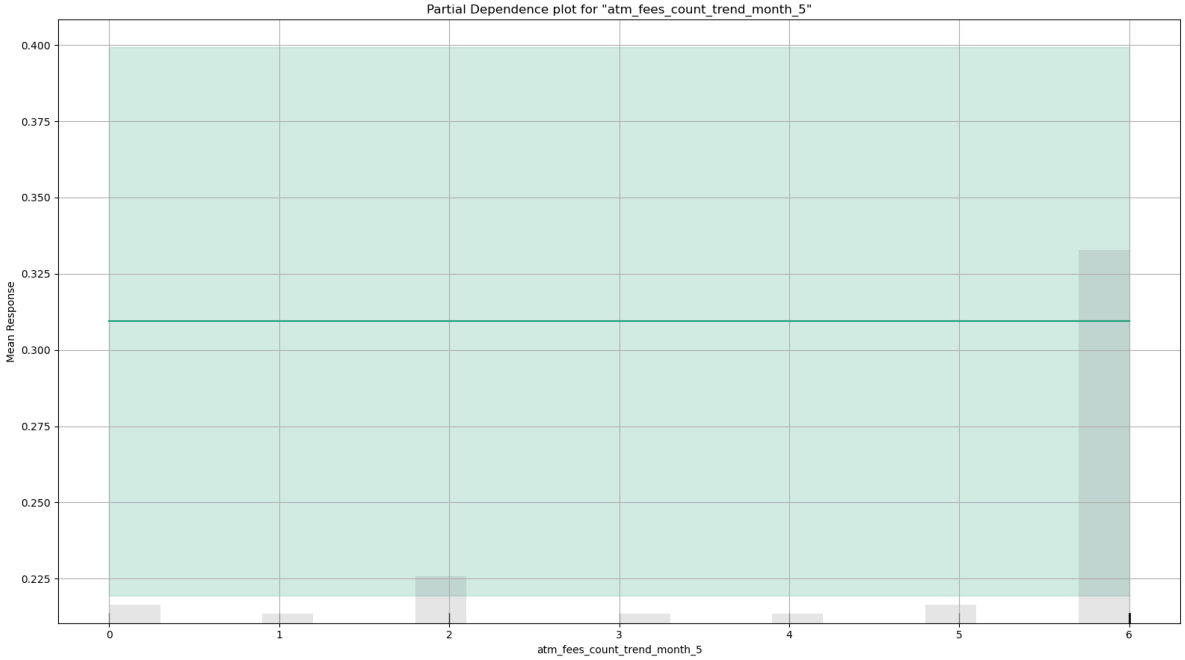
Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

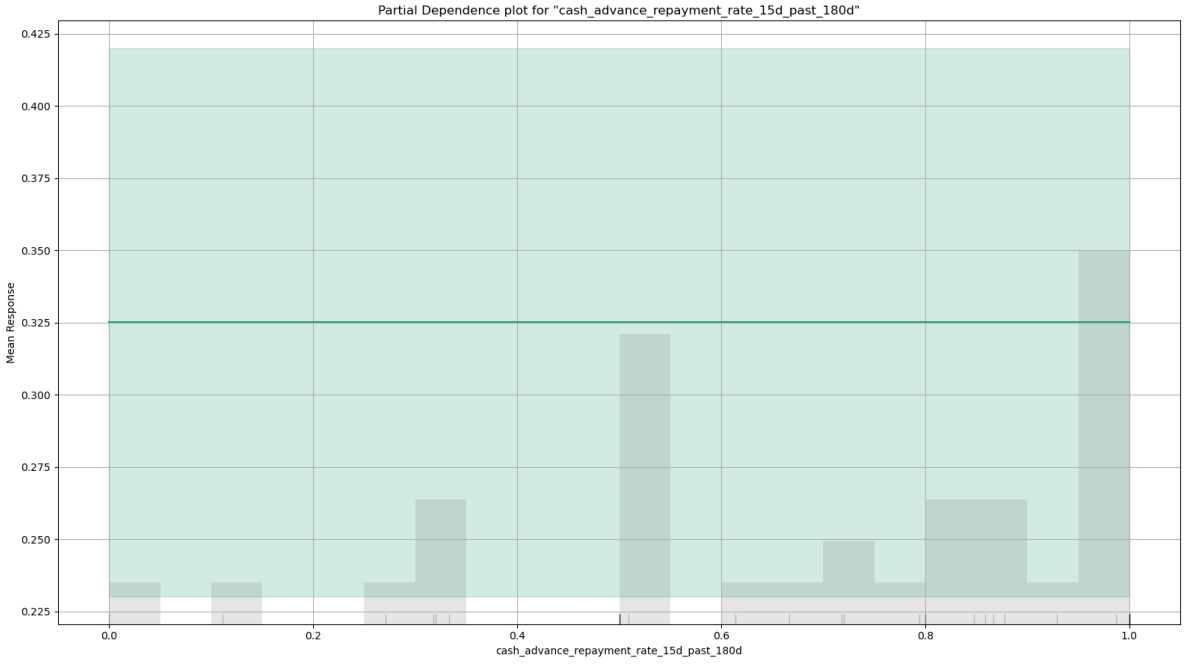

Partial Dependence plot for "primary_account_current_balance_maximum_past_180d"
Mean Response
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
−1000
−500
0
500
1000
1500
2000
2500
primary_account_current_balance_maximum_past_180d


Partial Dependence plot for "primary_account_current_balance_average_past_30d"
Mean Response
0.40
0.35
0.30
0.25
0.20
−1000
−500
0
500
1000
1500
2000
2500
primary_account_current_balance_average_past_30d


Partial Dependence plot for "primary_account_history_days"
Mean Response
0.40
0.35
0.30
0.25
100
200
300
400
500
600
700
primary_account_history_days

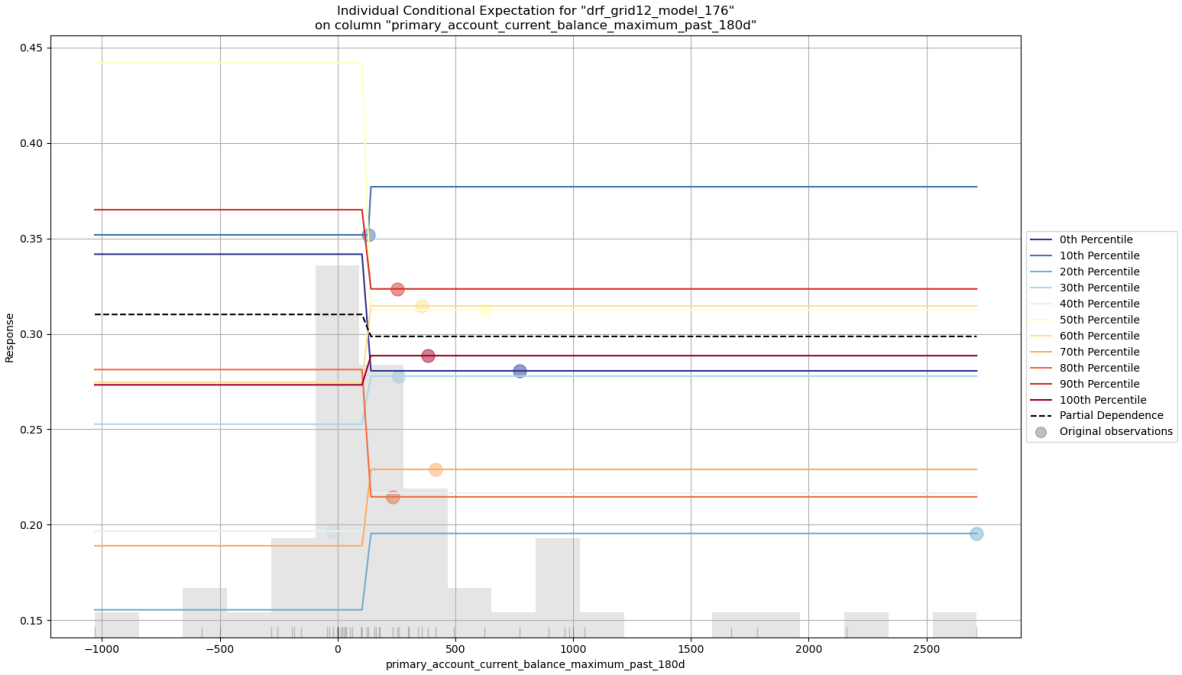

Partial Dependence plot for "minimum_current_balance_2d_after_payroll_past_180d"
Mean Response
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
−1500
−1000
−500
0
500
1000
1500
minimum_current_balance_2d_after_payroll_past_180d

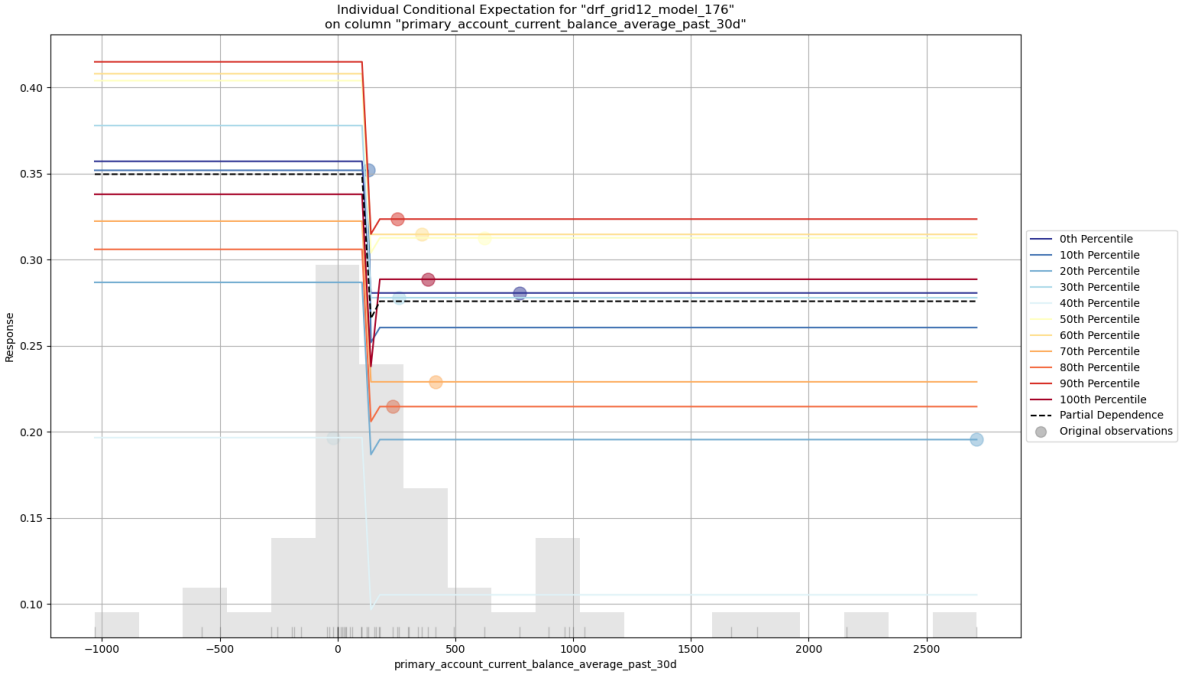

Partial Dependence plot for "atm_fees_count_trend_month_5"
Mean Response
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0
1
2
3
4
5
6
atm_fees_count_trend_month_5

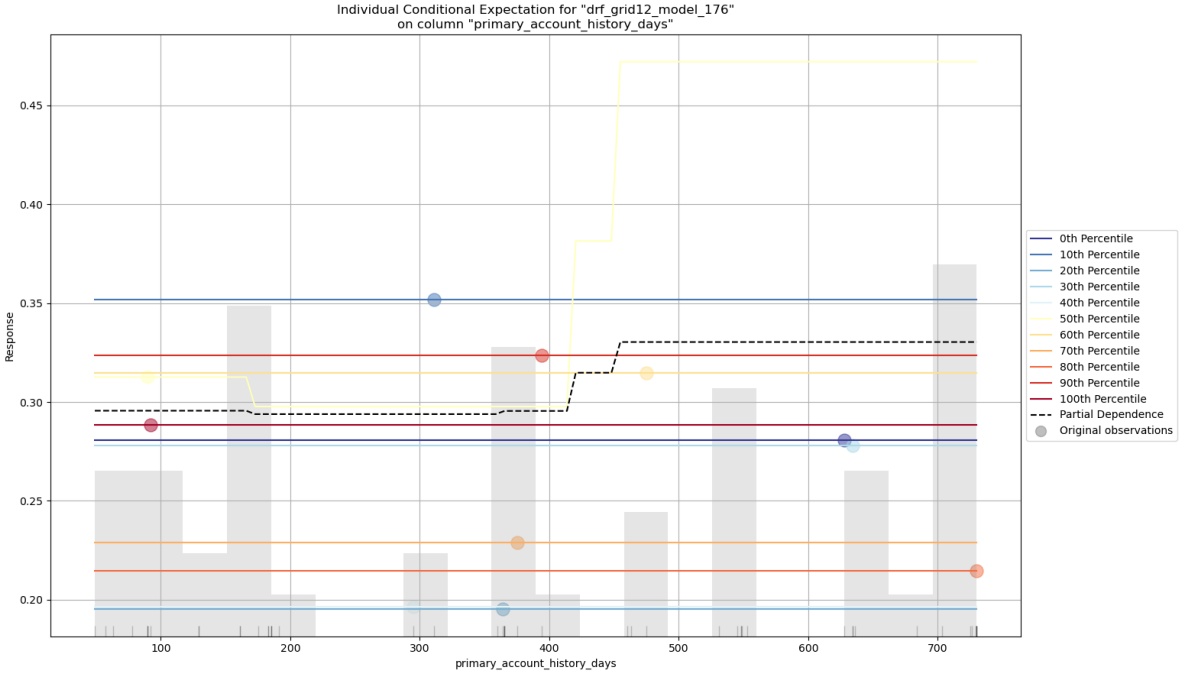

Partial Dependence plot for "cash_advance_repayment_rate_15d_past_180d"
Mean Response
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.0
0.2
0.4
0.6
0.8
1.0
cash_advance_repayment_rate_15d_past_180d

Individual Conditional Expectation

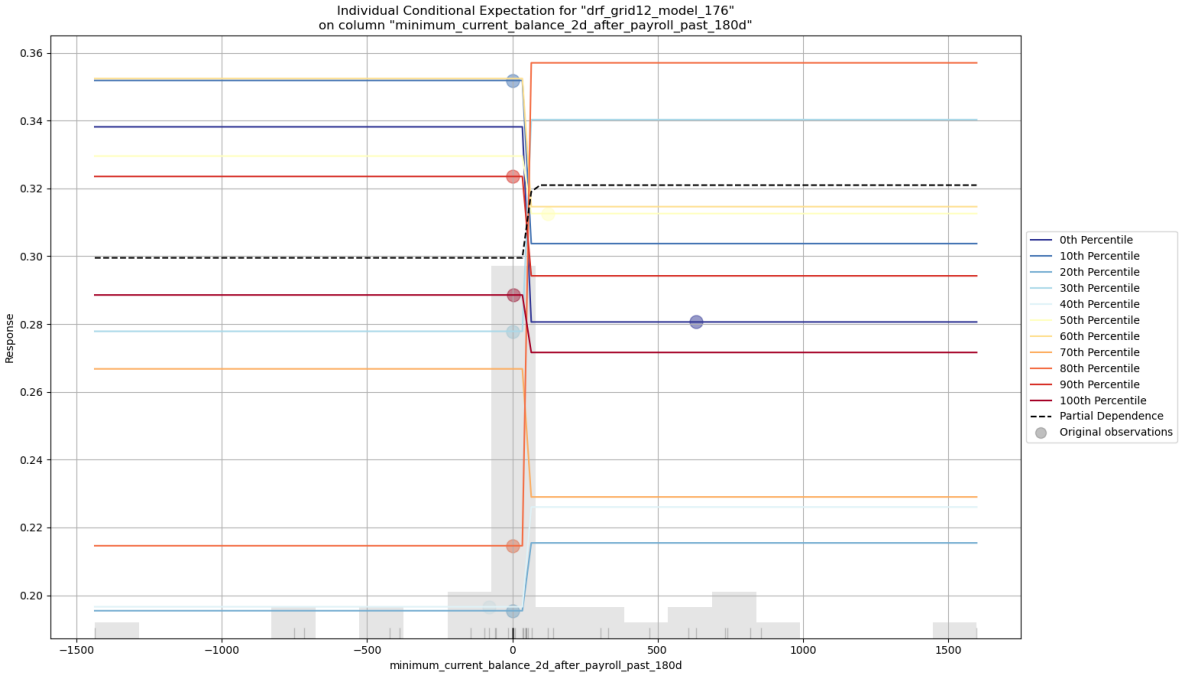
An Individual Conditional Expectation (ICE) plot gives a graphical depiction of the marginal effect of a variable on the response. ICE plots are similar to partial dependence plots (PDP); PDP shows the average effect of a feature while ICE plot shows the effect for a single instance. This function will plot the effect for each decile. In contrast to the PDP, ICE plots can provide more insight, especially when there is stronger feature interaction.

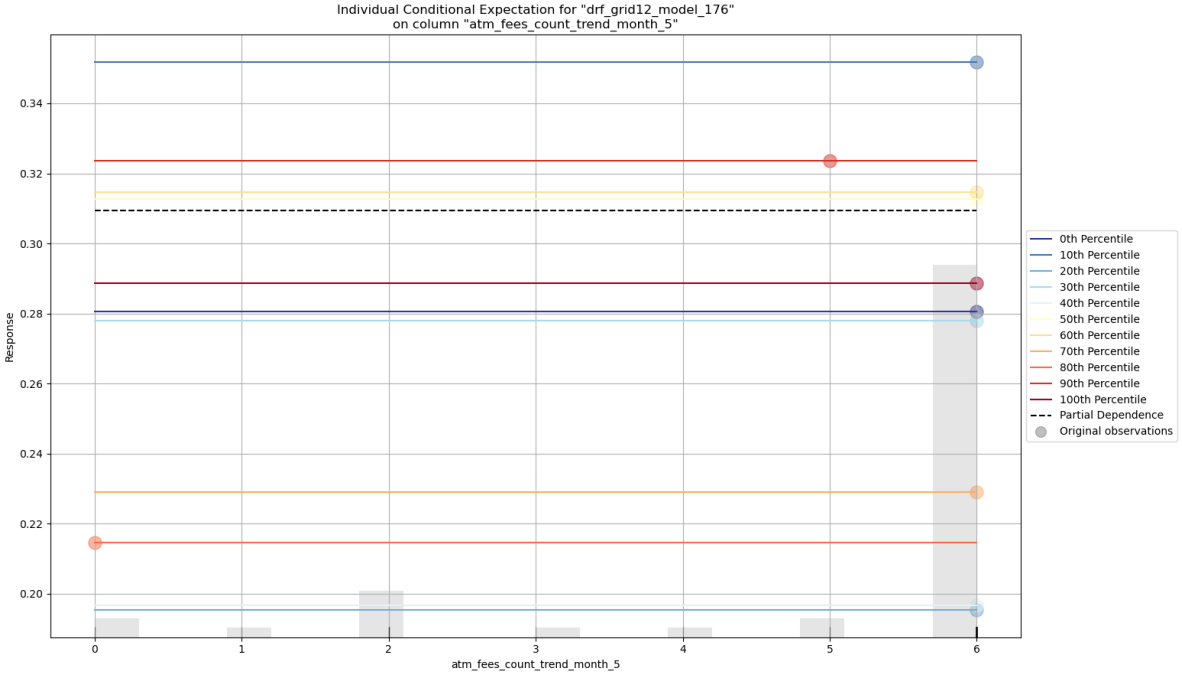



Individual Conditional Expectation for "drf_grid12_model_176"
on column "primary_account_current_balance_average_past_30d"
Response
primary_account_current_balance_average_past_30d
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations

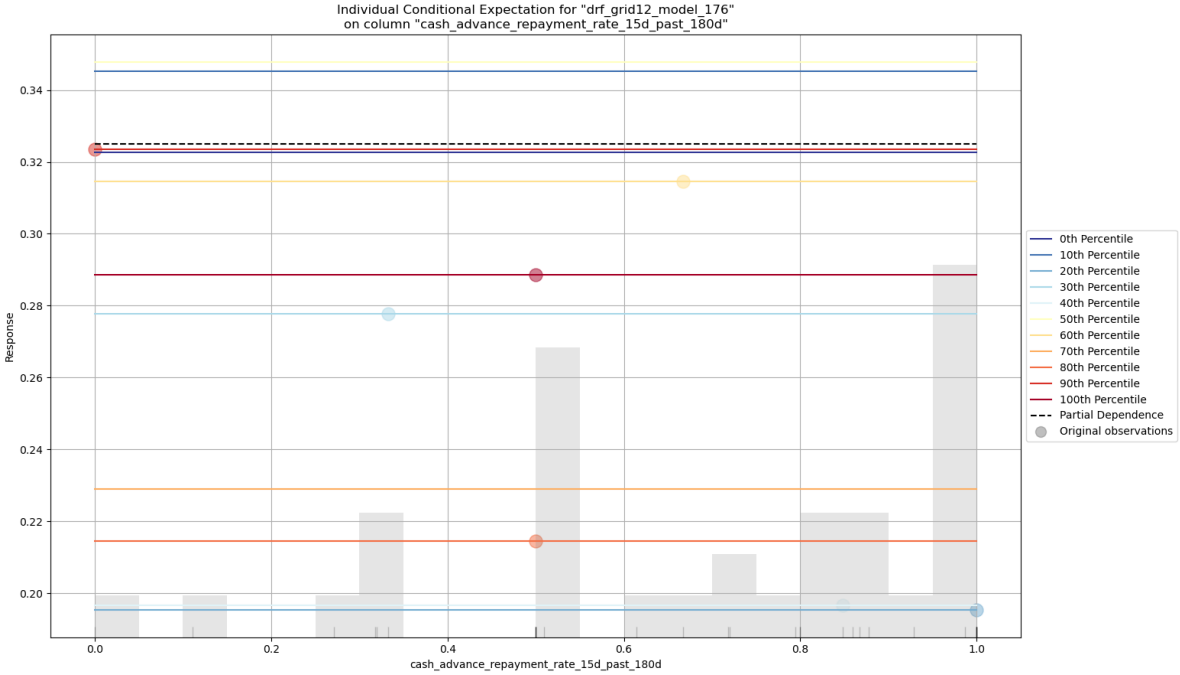

Individual Conditional Expectation for "drf_grid12_model_176"
on column "primary_account_history_days"
Response
primary_account_history_days
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations

```
/opt/anaconda3/lib/python3.12/site-packages/h2o/explanation/_explain.py:1728: UserWarning: Or
  warnings.warn(msg)
/opt/anaconda3/lib/python3.12/site-packages/h2o/explanation/_explain.py:1728: UserWarning: Or
  warnings.warn(msg)
```



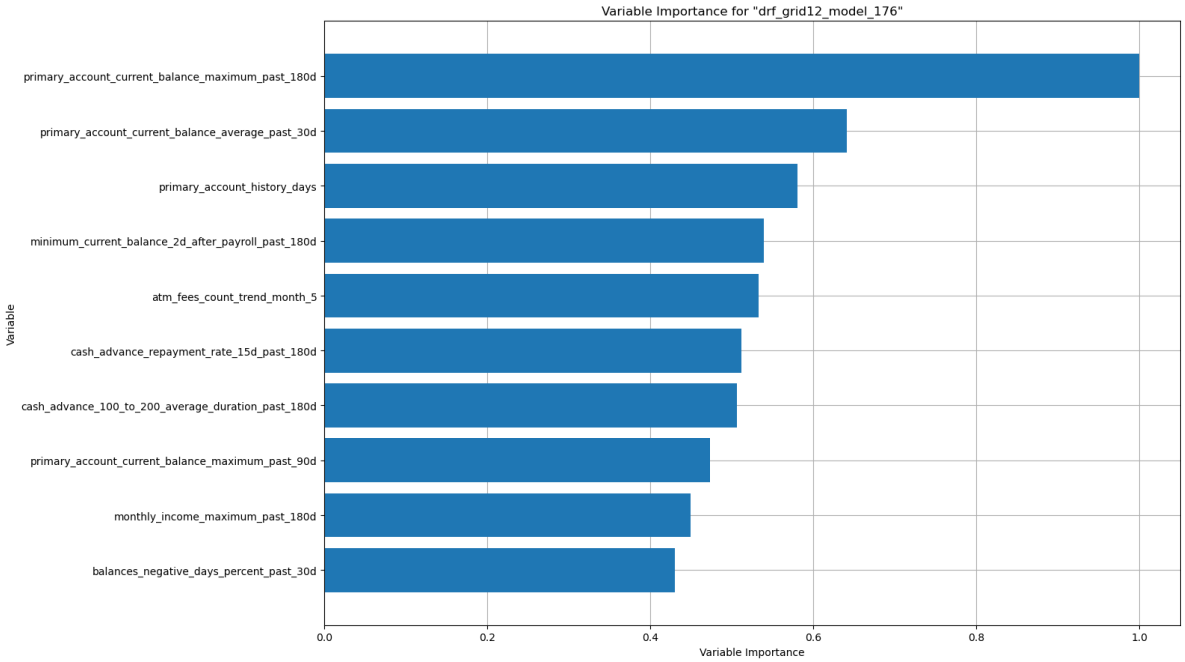



```
/opt/anaconda3/lib/python3.12/site-packages/h2o/explanation/_explain.py:1728: UserWarning: Or
  warnings.warn(msg)
/opt/anaconda3/lib/python3.12/site-packages/h2o/explanation/_explain.py:1728: UserWarning: Or
  warnings.warn(msg)
/opt/anaconda3/lib/python3.12/site-packages/h2o/explanation/_explain.py:1728: UserWarning: Or
  warnings.warn(msg)
/opt/anaconda3/lib/python3.12/site-packages/h2o/explanation/_explain.py:1728: UserWarning: Or
  warnings.warn(msg)
```



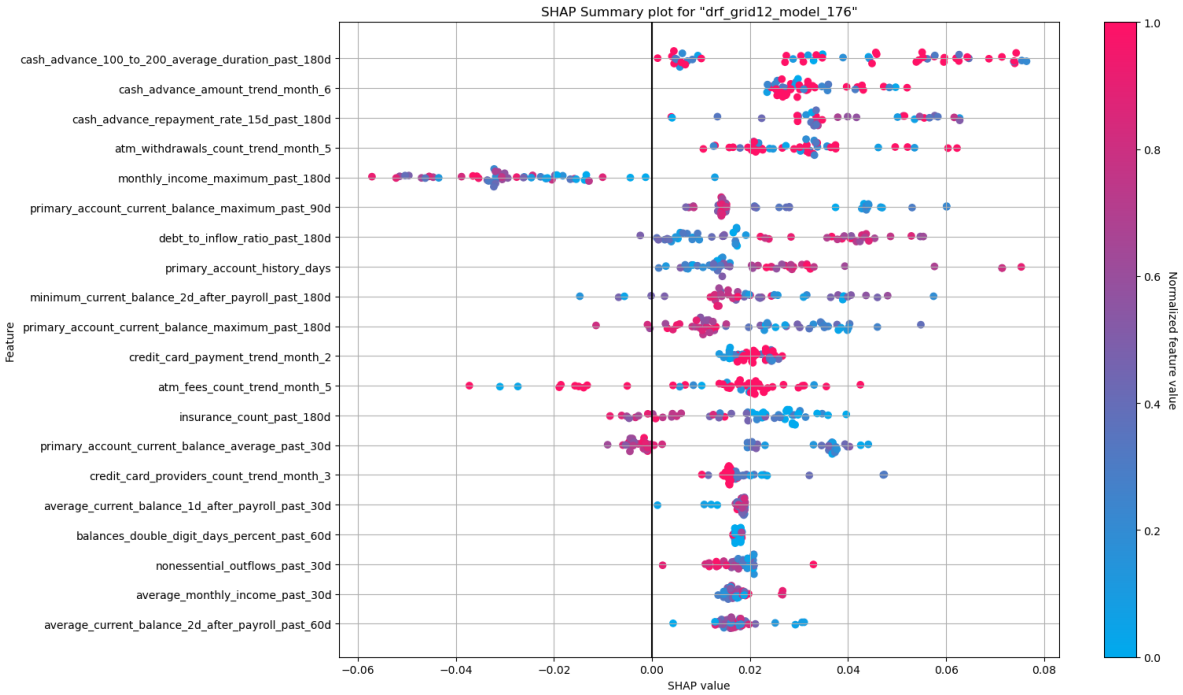
Variable Importance

The variable importance plot shows the relative importance of the most important variables in the model.



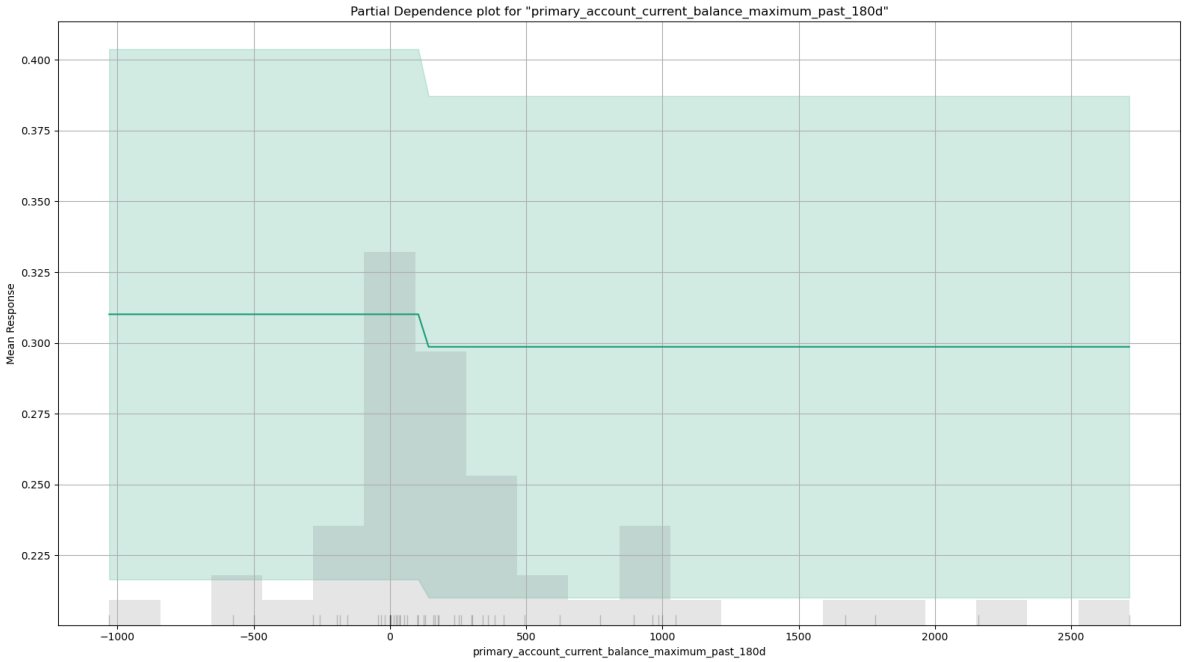
SHAP Summary

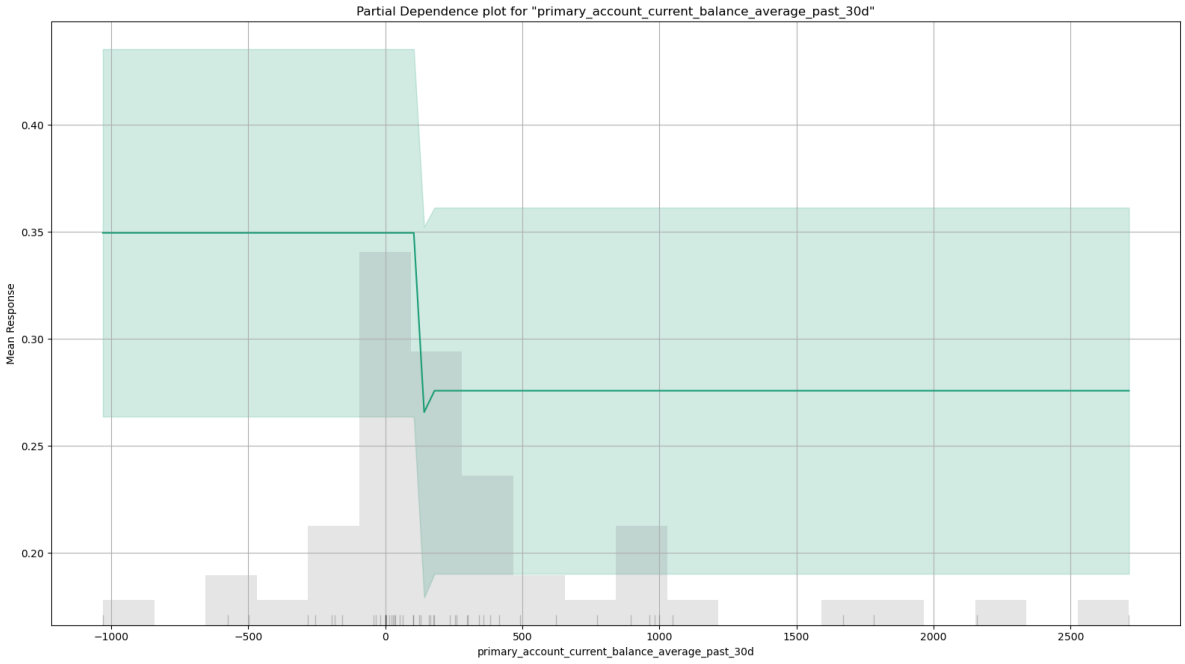
SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.

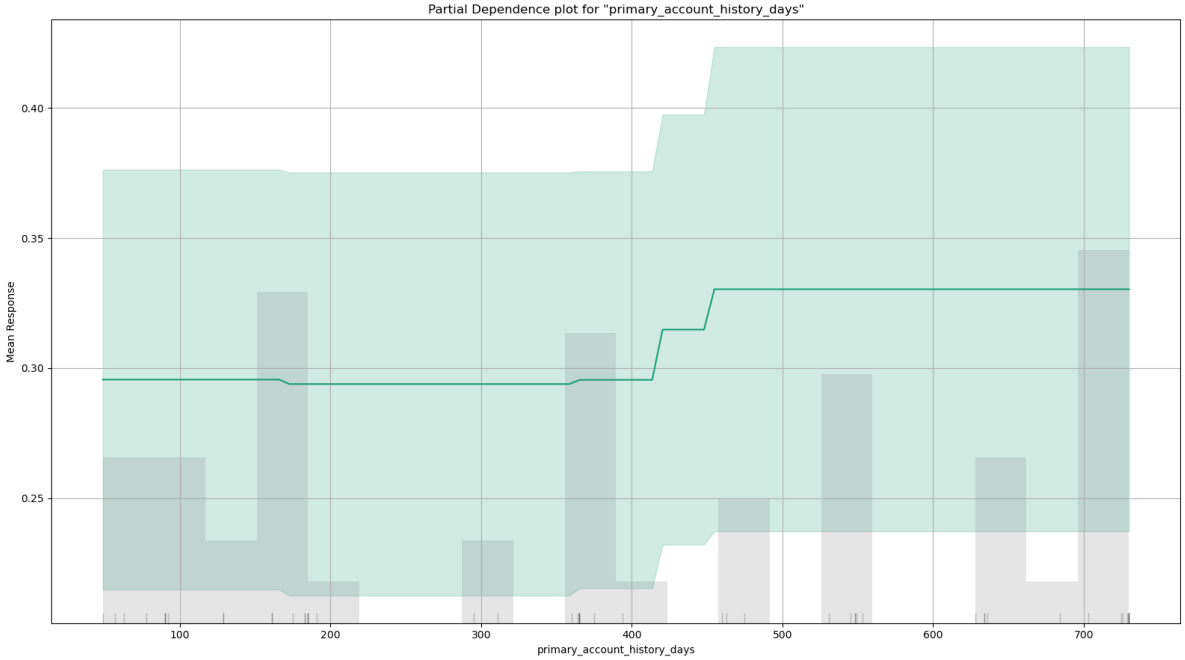


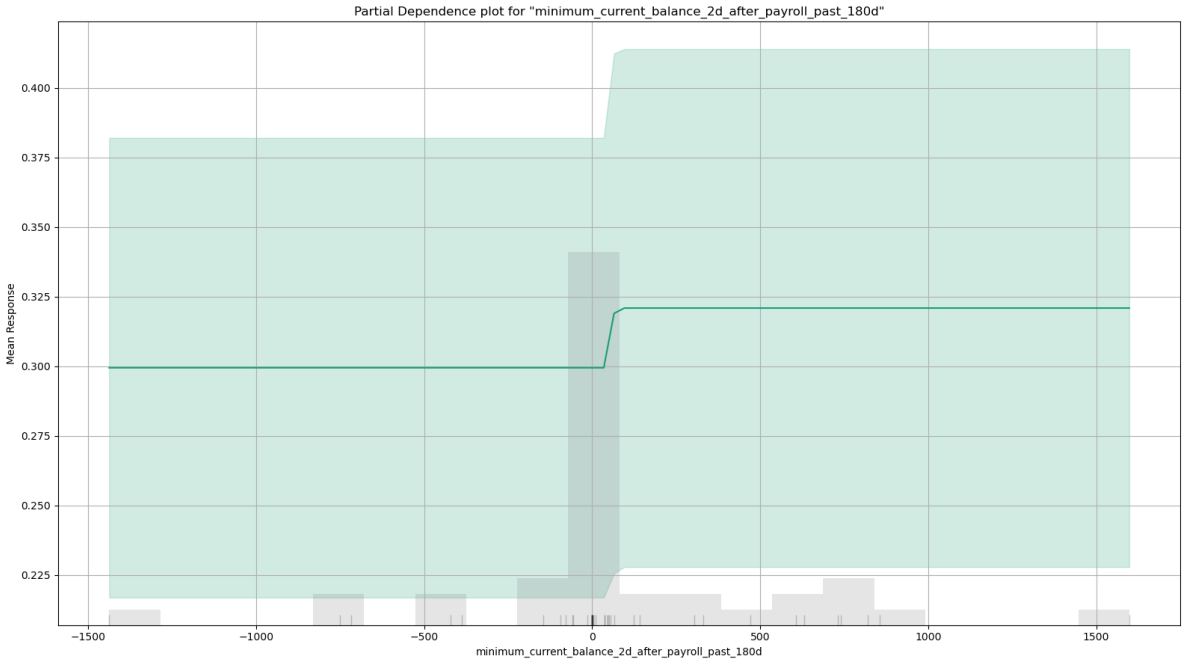
Partial Dependence Plots

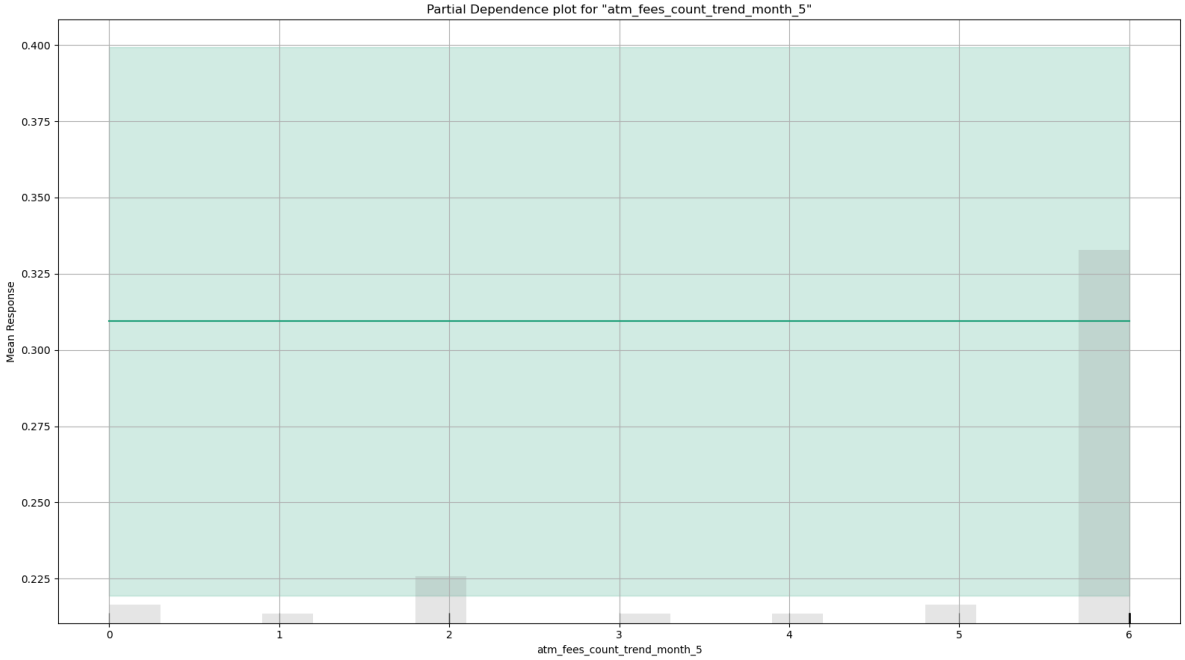
Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

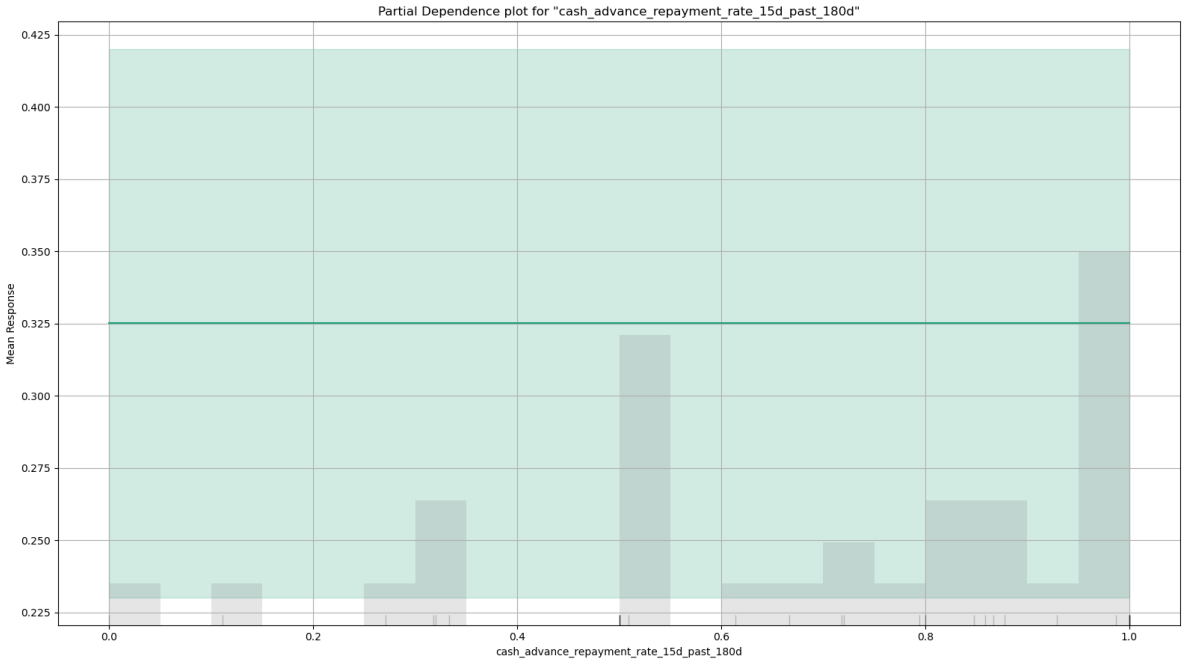

Partial Dependence plot for "primary_account_current_balance_maximum_past_180d"
Mean Response
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
−1000
−500
0
500
1000
1500
2000
2500
primary_account_current_balance_maximum_past_180d


Partial Dependence plot for "primary_account_current_balance_average_past_30d"
Mean Response
0.40
0.35
0.30
0.25
0.20
−1000
−500
0
500
1000
1500
2000
2500
primary_account_current_balance_average_past_30d


Partial Dependence plot for "primary_account_history_days"
Mean Response
0.40
0.35
0.30
0.25
100
200
300
400
500
600
700
primary_account_history_days

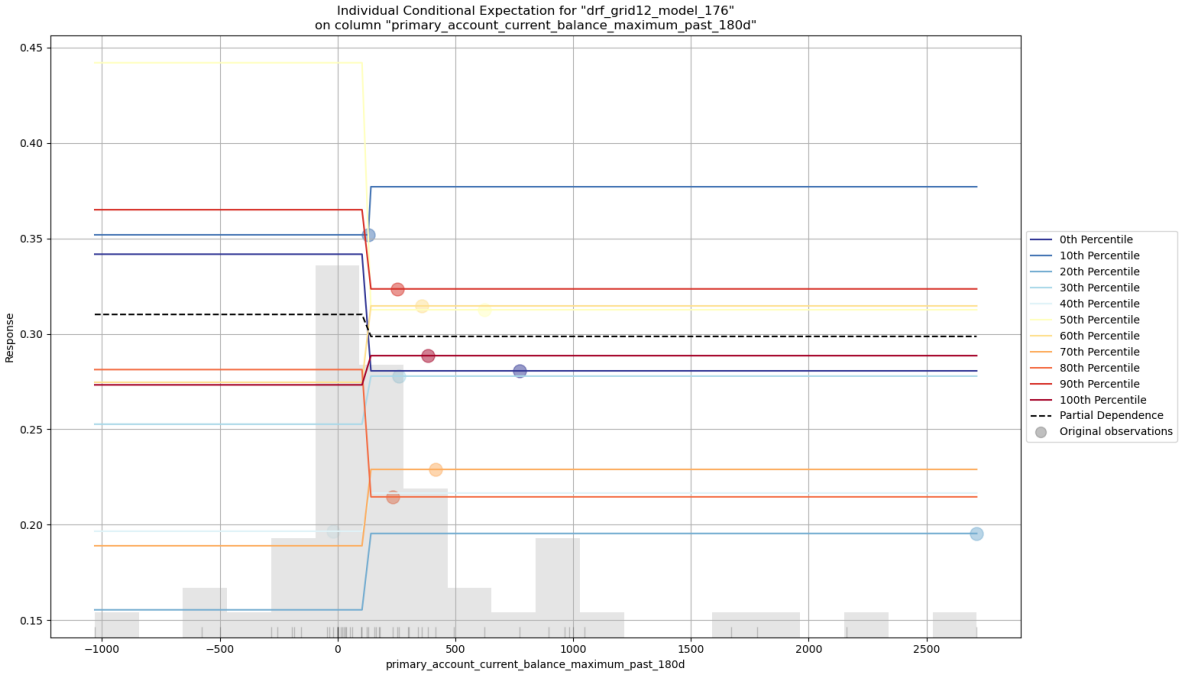

Partial Dependence plot for "minimum_current_balance_2d_after_payroll_past_180d"
Mean Response
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
−1500
−1000
−500
0
500
1000
1500
minimum_current_balance_2d_after_payroll_past_180d

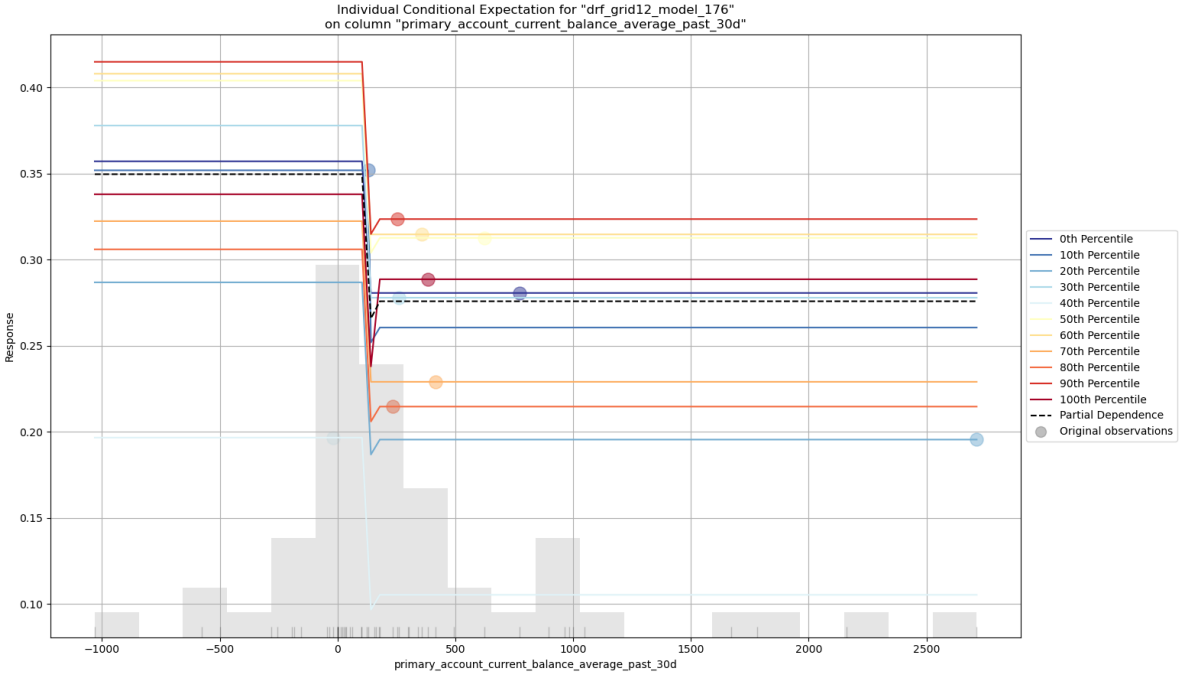

Partial Dependence plot for "atm_fees_count_trend_month_5"
Mean Response
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0
1
2
3
4
5
6
atm_fees_count_trend_month_5

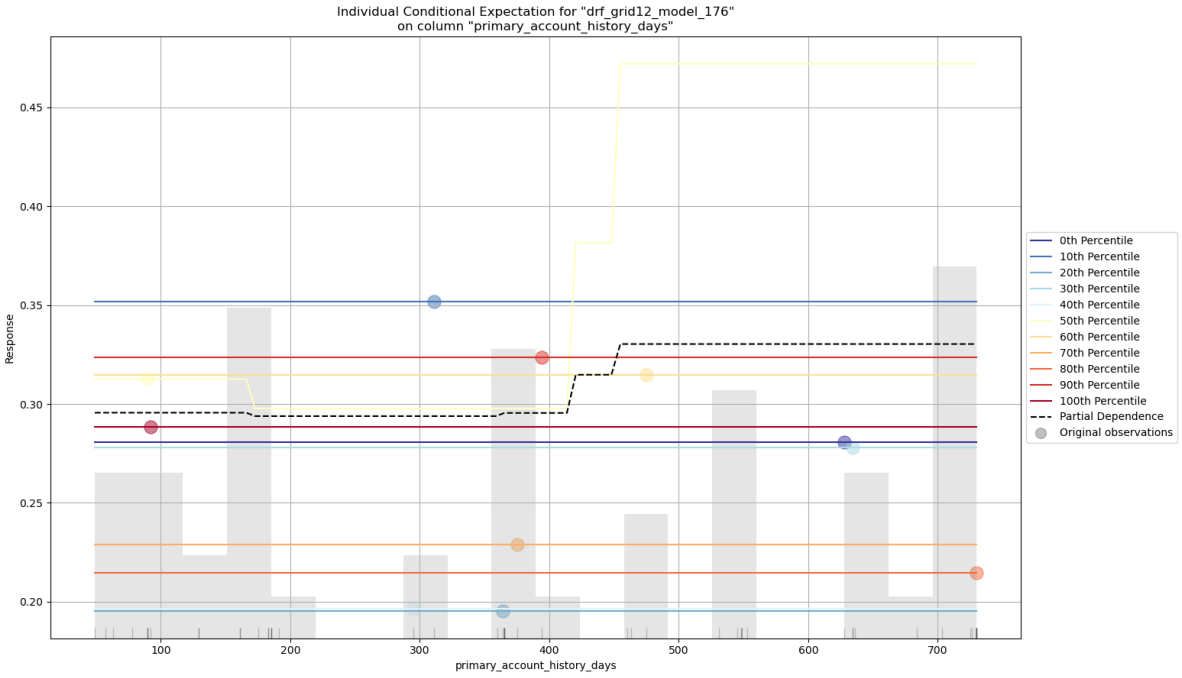

Partial Dependence plot for "cash_advance_repayment_rate_15d_past_180d"
Mean Response
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.0
0.2
0.4
0.6
0.8
1.0
cash_advance_repayment_rate_15d_past_180d

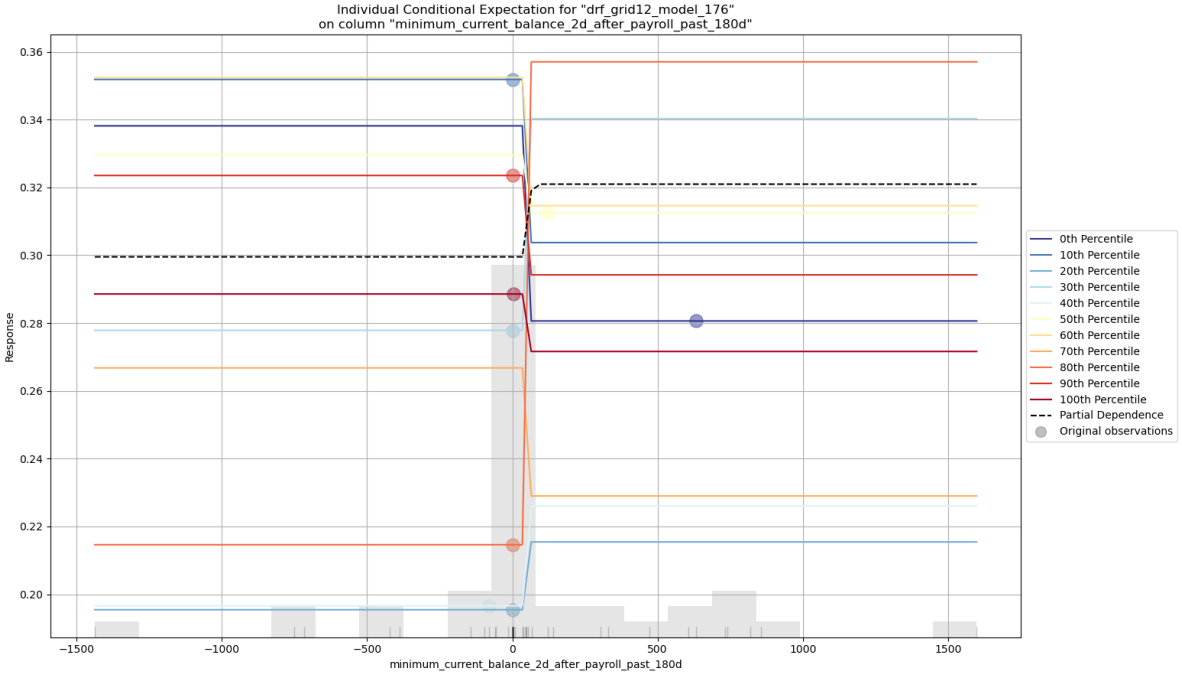
Individual Conditional Expectation

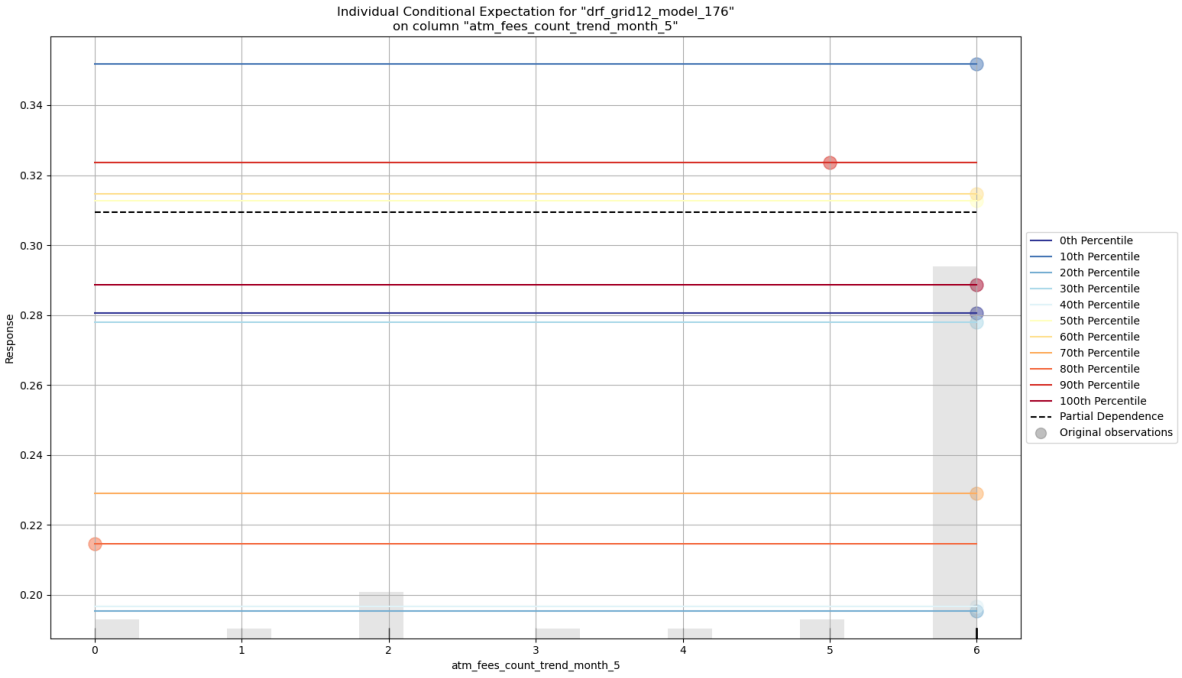
An Individual Conditional Expectation (ICE) plot gives a graphical depiction of the marginal effect of a variable on the response. ICE plots are similar to partial dependence plots (PDP); PDP shows the average effect of a feature while ICE plot shows the effect for a single instance. This function will plot the effect for each decile. In contrast to the PDP, ICE plots can provide more insight, especially when there is stronger feature interaction.

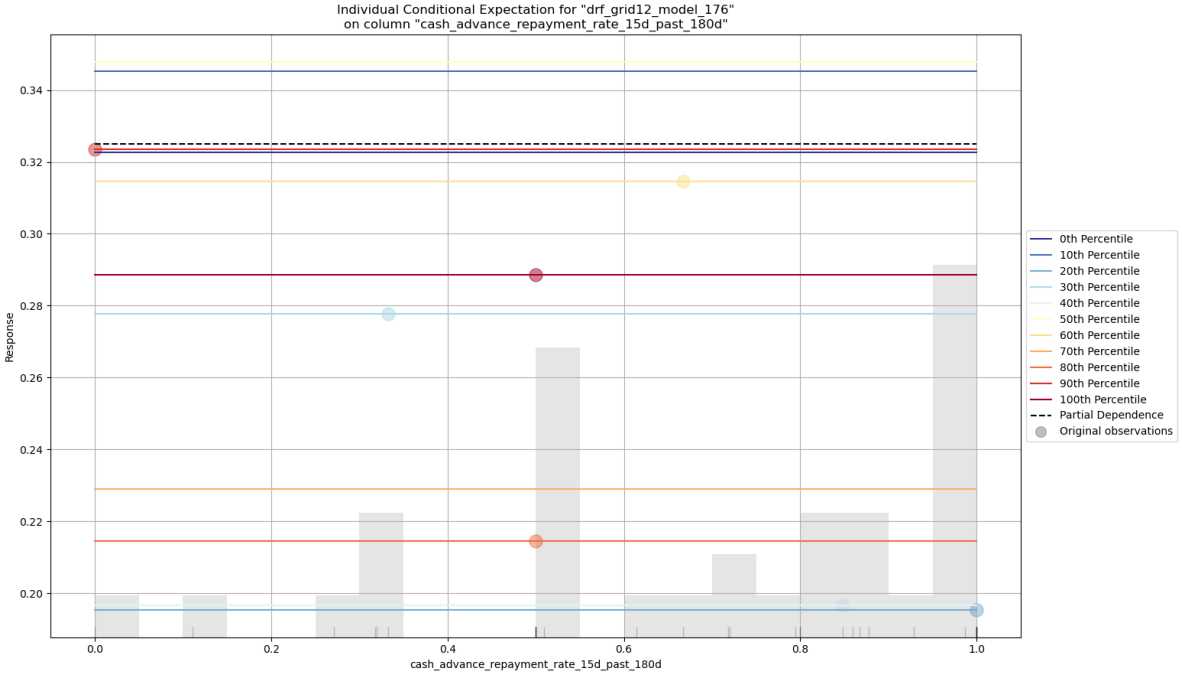



Individual Conditional Expectation for "drf_grid12_model_176"
on column "primary_account_current_balance_average_past_30d"
Response
primary_account_current_balance_average_past_30d
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations


Individual Conditional Expectation for "drf_grid12_model_176"
on column "primary_account_history_days"
Response
primary_account_history_days
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations

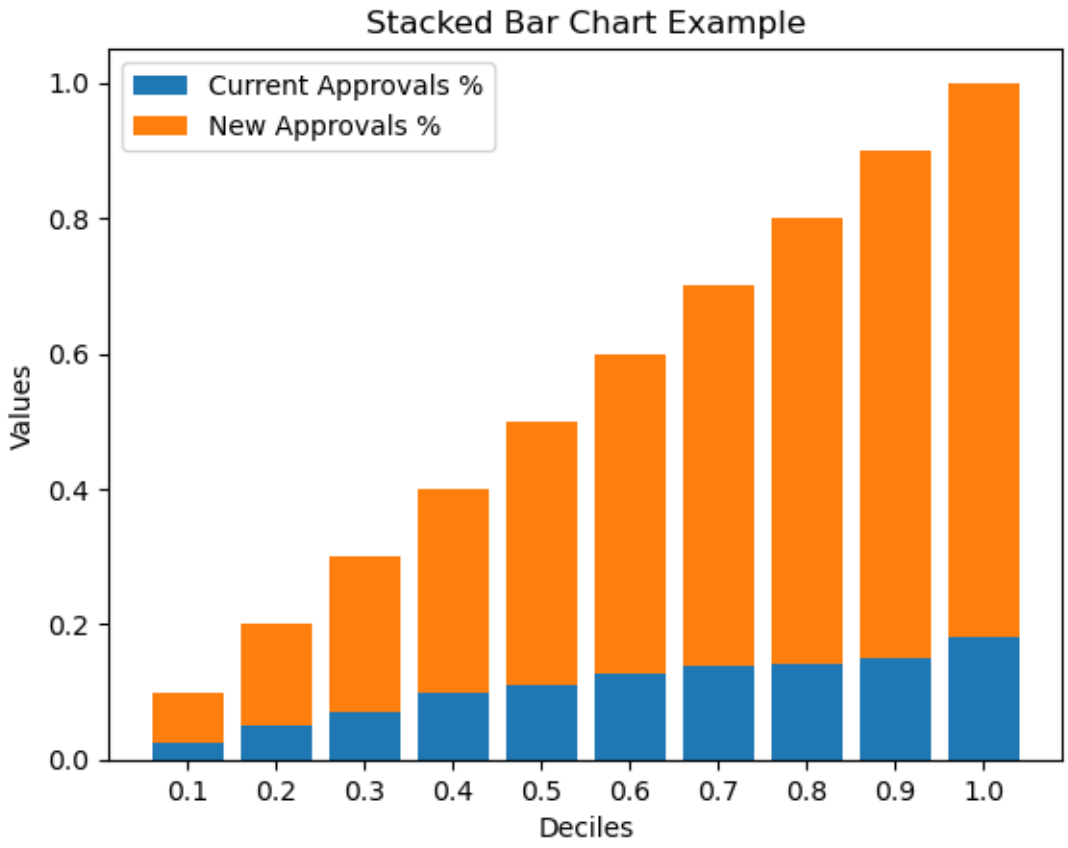

Individual Conditional Expectation for "drf_grid12_model_176"
on column "minimum_current_balance_2d_after_payroll_past_180d"
Response
minimum_current_balance_2d_after_payroll_past_180d
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations


Individual Conditional Expectation for "drf_grid12_model_176"
on column "atm_fees_count_trend_month_5"
Response
atm_fees_count_trend_month_5
0th Percentile
10th Percentile
20th Percentile
30th Percentile
40th Percentile
50th Percentile
60th Percentile
70th Percentile
80th Percentile
90th Percentile
100th Percentile
Partial Dependence
Original observations




```
import matplotlib.pyplot as plt
import numpy as np

# Sample data
categories = ['0.1', '0.2', '0.3', '0.4', '0.5',
              '0.6', '0.7', '0.8', '0.9', '1.0']


values1 = [0.026,
0.049,
0.070,
0.099,
0.111,
0.126,
0.137,
0.141,
0.150,
0.180]

values2 = [0.074,
0.151,
0.230,
0.301,
```

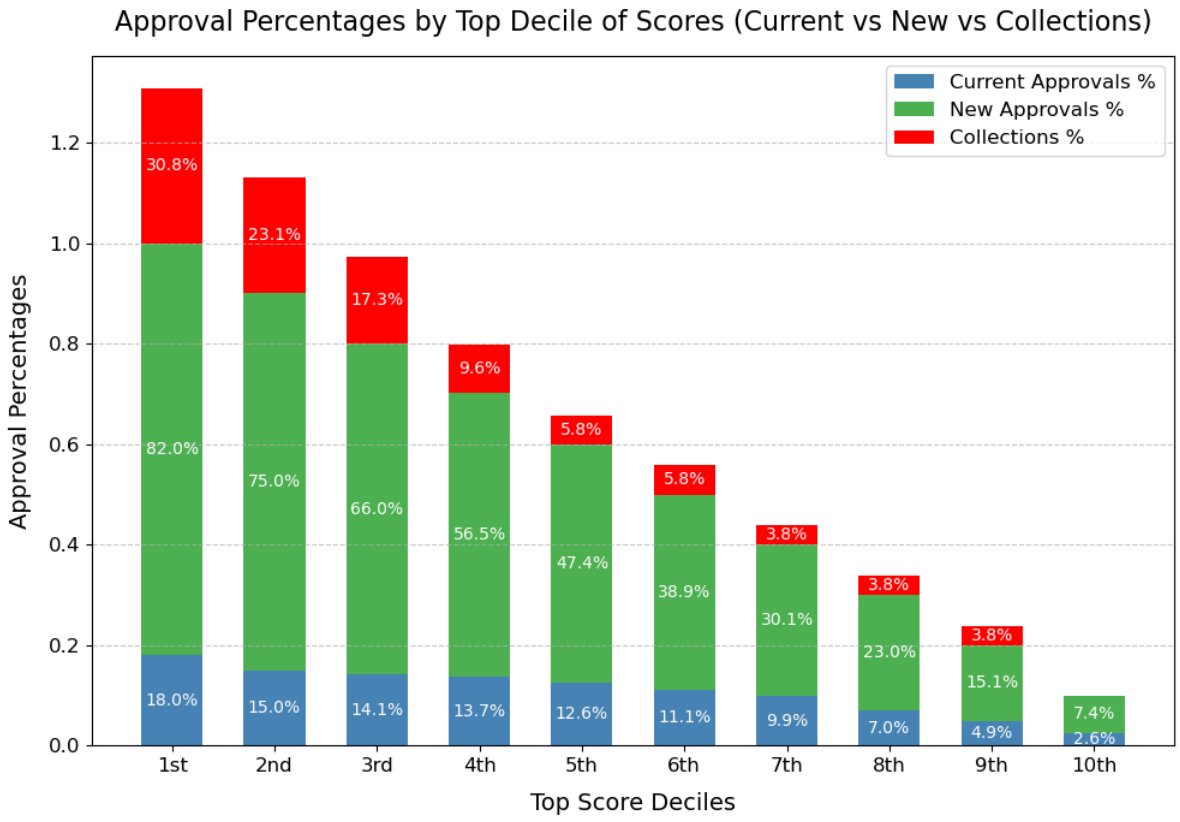
```
0.389,
0.474,
0.565,
0.660,
0.750,
0.820]



# Set the positions
x = np.arange(len(categories))  # The label locations

# Plotting the stacked bars
plt.bar(x, values1, label='Current Approvals %')
plt.bar(x, values2, bottom=values1, label='New Approvals %')  # Stack values2 on top of value

# Adding labels and title
plt.xlabel('Deciles')
plt.ylabel('Values')
plt.title('Stacked Bar Chart Example')
plt.xticks(x, categories)  # Set the x-axis labels

# Adding legend
plt.legend()

# Show the plot
plt.show()
```




```
values3 = [0.0,
0.0384615,
0.0384615,
0.0384615,
0.0576923,
0.0576923,
0.0961538,
0.173077,
0.230769,
0.307692]
```

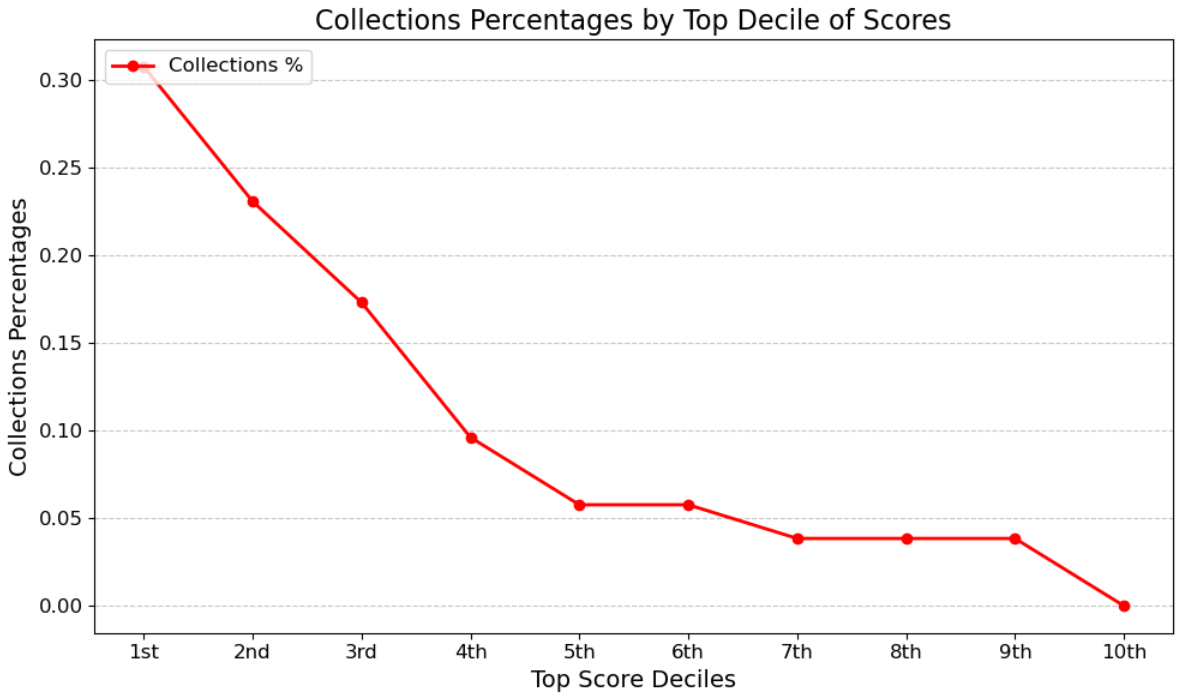
```
import matplotlib.pyplot as plt
import numpy as np

# Updated sample data for categories and values
categories = ['1st', '2nd', '3rd', '4th', '5th', '6th', '7th', '8th', '9th', '10th']
values1 = np.flip([0.026, 0.049, 0.070, 0.099, 0.111, 0.126, 0.137, 0.141, 0.150, 0.180])
```

```
values2 = np.flip([0.074, 0.151, 0.230, 0.301, 0.389, 0.474, 0.565, 0.660, 0.750, 0.820])
values3 = np.flip([0.0, 0.0384615, 0.0384615, 0.0384615, 0.0576923, 0.0576923, 0.0961538, 0.1

# Set the positions
x = np.arange(len(categories))  # The label locations
bar_width = 0.6

# Plotting the stacked bars with custom colors
plt.figure(figsize=(10,7))
bars1 = plt.bar(x, values1, color="#4682B4", width=bar_width, label='Current Approvals %')  #
bars2 = plt.bar(x, values2, bottom=values1, color="#4CAF50", width=bar_width, label='New Appr
bars3 = plt.bar(x, values3, bottom=np.add(values1, values2), color="#FF0000", width=bar_width

# Adding labels and title with custom fonts
plt.xlabel('Top Score Deciles', fontsize=14, labelpad=10)
plt.ylabel('Approval Percentages', fontsize=14, labelpad=10)
plt.title('Approval Percentages by Top Decile of Scores (Current vs New vs Collections)', for

# Customize x-axis ticks and grid
plt.xticks(x, categories, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adding legend with custom location and fontsize
plt.legend(loc='upper right', fontsize=12)

# Adding percentage labels to each bar segment, excluding the 1st decile for Collections %
for i in range(len(categories)):
    # Current Approvals %
    plt.text(x[i], values1[i] / 2, f"{values1[i]:.1%}", ha='center', va='center', color="whit
    # New Approvals %
    plt.text(x[i], values1[i] + values2[i] / 2, f"{values2[i]:.1%}", ha='center', va='center
    # Additional Approvals % - only add label if not the first decile
    if i != 9:
        plt.text(x[i], values1[i] + values2[i] + values3[i] / 2, f"{values3[i]:.1%}", ha='cen

# Show the plot
plt.tight_layout()  # Adjusts layout for better spacing
plt.show()
```



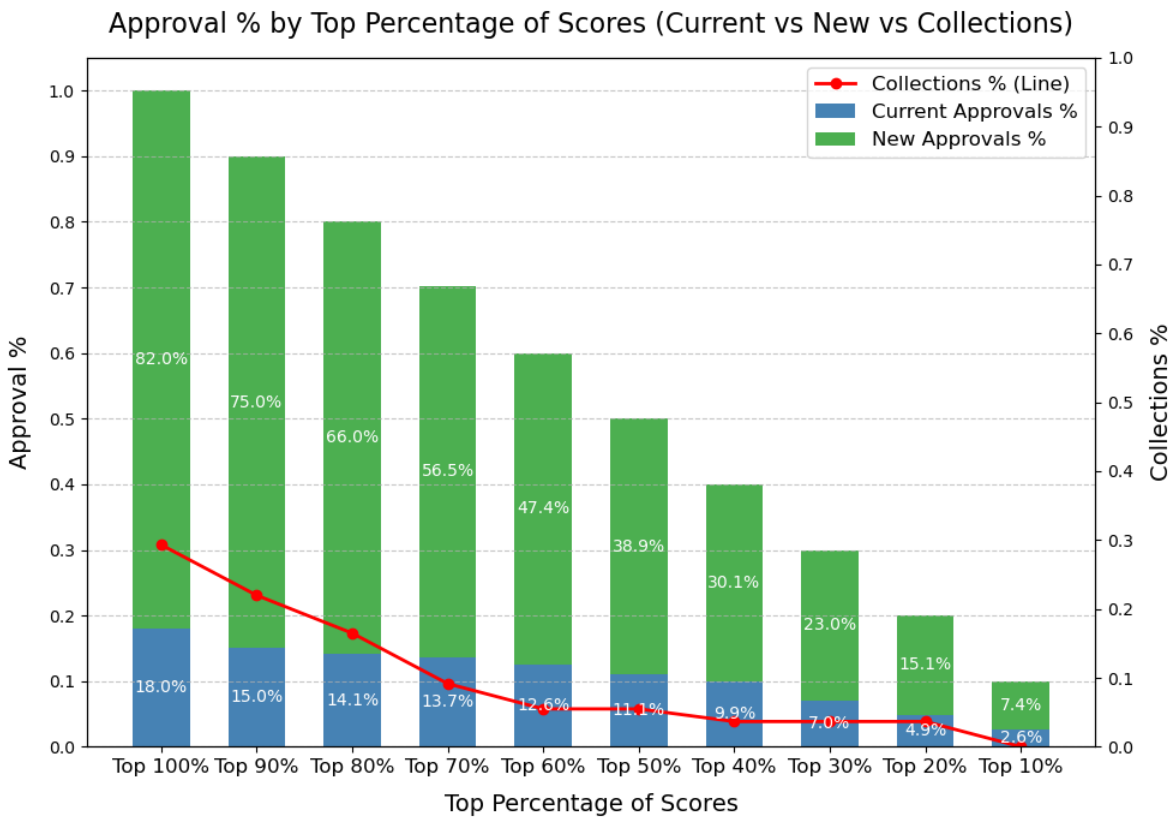

```
import matplotlib.pyplot as plt
import numpy as np

# Updated sample data for categories and collections values
categories = ['1st', '2nd', '3rd', '4th', '5th', '6th', '7th', '8th', '9th', '10th']
values3 = np.flip([0.0, 0.0384615, 0.0384615, 0.0384615, 0.0576923, 0.0576923, 0.0961538, 0.1

# Set up the line plot
plt.figure(figsize=(10, 6))
plt.plot(categories, values3, color="#FF0000", marker='o', linestyle='-', linewidth=2, marker

# Adding labels and title with custom fonts
plt.xlabel('Top Score Deciles', fontsize=14)
plt.ylabel('Collections Percentages', fontsize=14)
plt.title('Collections Percentages by Top Decile of Scores', fontsize=16)

# Customize x-axis and y-axis ticks and grid
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adding legend
plt.legend(loc='upper left', fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

# Updated sample data for categories and values
categories = ['Top 100%', 'Top 90%', 'Top 80%', 'Top 70%', 'Top 60%', 'Top 50%', 'Top 40%',
values1 = np.flip([0.026, 0.049, 0.070, 0.099, 0.111, 0.126, 0.137, 0.141, 0.150, 0.180])
values2 = np.flip([0.074, 0.151, 0.230, 0.301, 0.389, 0.474, 0.565, 0.660, 0.750, 0.820])
values3 = np.flip([0.0, 0.0384615, 0.0384615, 0.0384615, 0.0576923, 0.0576923, 0.0961538, 0.

# Set the positions
x = np.arange(len(categories))  # The label locations
bar_width = 0.6
```

```
# Create figure and axis objects
fig, ax1 = plt.subplots(figsize=(10, 7))

# Plotting the stacked bars for Current and New Approvals
bars1 = ax1.bar(x, values1, color="#4682B4", width=bar_width, label='Current Approvals %')  #
bars2 = ax1.bar(x, values2, bottom=values1, color="#4CAF50", width=bar_width, label='New Appr

# Adding the line plot for Collections % on top of the bars
ax1.plot(x, values3, color="#FF0000", marker='o', linestyle='-', linewidth=2, markersize=6, l

# Adding labels and title with custom fonts
ax1.set_xlabel('Top Percentage of Scores', fontsize=14, labelpad=10)
ax1.set_ylabel('Approval %', fontsize=14, labelpad=10)
ax1.set_title('Approval % by Top Percentage of Scores (Current vs New vs Collections)', fonts

# Customize x-axis ticks
ax1.set_xticks(x)
ax1.set_xticklabels(categories, fontsize=12)

# Set y-axis ticks and grid at intervals of 0.1
ax1.set_yticks(np.arange(0, 1.1, 0.1))  # Set y-ticks from 0 to 1 in increments of 0.1
ax1.grid(axis='y', linestyle='--', alpha=0.7)

# Adding secondary y-axis for Collections %
ax2 = ax1.twinx()  # Creates a secondary y-axis sharing the same x-axis
ax2.set_ylabel('Collections %', fontsize=14, labelpad=10)
ax2.set_yticks(np.arange(0, 1.1, 0.1))  # Set same tick intervals for consistency
ax2.tick_params(axis='y')  # Set tick color for distinction

# Adding legend with custom location and fontsize
ax1.legend(loc='upper right', fontsize=12)

# Adding percentage labels to each bar segment
for i in range(len(categories)):
    # Current Approvals %
    ax1.text(x[i], values1[i] / 2, f"{values1[i]:.1%}", ha='center', va='center', color="whi
    # New Approvals %
    ax1.text(x[i], values1[i] + values2[i] / 2, f"{values2[i]:.1%}", ha='center', va='center

# Show the plot
fig.tight_layout()  # Adjusts layout for better spacing
plt.show()
```



Create Customer file

```
df = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Visible_1169_Records_Scored_2024
print(df.shape)
```

```
(1169, 3442)
```

```
/var/folders/z5/xxddnyr978bb97d94x1wj3780000gn/T/ipykernel_92556/3848255160.py:1: DtypeWarni
  df = pd.read_csv("/Users/luis/Evaluations/Visible/3_Results/Visible_1169_Records_Scored_20
```

```
view = df[0:1].transpose()
view = view.reset_index(drop = False)
view
```